
beam Documentation

Release 0.7.0

The BEAM Team

Jun 24, 2019

| | | |
|----------|---|-----------|
| 1 | About BEAM | 3 |
| 1.1 | Overview | 3 |
| 1.2 | MATSim Integration | 4 |
| 1.2.1 | BeamMobSim | 4 |
| 1.2.2 | AgentSim | 5 |
| 1.2.3 | PhysSim | 5 |
| 1.2.4 | R5 Router | 5 |
| 1.2.5 | MATSim Events | 6 |
| 1.3 | Resource Markets | 7 |
| 1.4 | Dynamic Within-Day Planning | 8 |
| 1.5 | Rich Modal Choice | 8 |
| 1.6 | Plug-in Electric Vehicle Modeling with BEAM | 8 |
| 1.7 | Contact Information | 8 |
| 1.8 | Reports and Papers | 9 |
| 1.9 | References | 9 |
| 2 | User's Guide | 11 |
| 2.1 | Getting Started | 11 |
| 2.1.1 | System Requirements | 11 |
| 2.1.2 | Prerequisites : | 11 |
| 2.1.3 | GIT-LFS Configuration | 12 |
| 2.1.4 | Installing BEAM | 12 |
| 2.1.5 | Running BEAM | 13 |
| 2.1.6 | Running BEAM with IntelliJ IDE | 13 |
| 2.1.7 | Scenarios | 15 |
| 2.1.8 | Inputs | 16 |
| 2.1.9 | Outputs | 17 |
| 2.1.10 | Model Config | 19 |
| 2.2 | Experiment Manager | 19 |
| 2.3 | Calibration | 21 |
| 2.3.1 | Optimization-based Calibration Principles | 21 |
| 2.3.2 | SigOpt Setup | 21 |
| 2.3.3 | Configuration | 22 |
| 2.3.4 | Execution | 23 |
| 2.3.5 | Manage Experiment | 23 |
| 2.4 | Timezones and GTFS | 23 |

| | | |
|----------|--|-----------|
| 2.5 | Converting a MATSim Scenario to Run with BEAM | 24 |
| 2.5.1 | Conversion Instructions | 24 |
| 3 | Developer's Guide | 27 |
| 3.1 | Repositories | 27 |
| 3.2 | Configuration | 28 |
| 3.3 | Environment Variables | 28 |
| 3.4 | GIT-LFS timeout - how to proceed | 29 |
| 3.5 | Keeping Production Data out of Master Branch | 29 |
| 3.6 | Automated Cloud Deployment | 30 |
| 3.6.1 | AWS EC2 Start | 31 |
| 3.6.2 | AWS EC2 Stop | 31 |
| 3.7 | Performance Monitoring | 31 |
| 3.7.1 | Beam Metrics Utility (<i>MetricsSupport</i>) | 31 |
| 3.7.2 | Beam Metrics Configuration | 32 |
| 3.7.3 | Setup Docker as Metric Backend | 33 |
| 3.8 | Tagging Tests for Periodic CI | 34 |
| 3.9 | Instructions for forking BEAM | 35 |
| 3.10 | Scala tips | 36 |
| 3.10.1 | Scala Collection | 36 |
| 3.10.2 | Use lazy logging | 38 |
| 4 | BeamAgents | 39 |
| 4.1 | Person Agents | 39 |
| 4.2 | Ride Hail Agents | 39 |
| 4.3 | Transit Driver Agents | 39 |
| 5 | Behaviors | 41 |
| 5.1 | Mode Choice | 41 |
| 5.1.1 | Multinomial Logit Mode Choice | 42 |
| 5.1.2 | Latent Class Mode Choice | 42 |
| 5.2 | Parking | 42 |
| 5.3 | Refueling | 43 |
| 6 | Event Specifications | 45 |
| 6.1 | MATSim Events | 45 |
| 6.1.1 | ActivityStartEvent | 45 |
| 6.1.2 | ActivityEndEvent | 45 |
| 6.1.3 | PersonDepartureEvent | 46 |
| 6.1.4 | PersonArrivalEvent | 46 |
| 6.1.5 | PersonEntersVehicleEvent | 46 |
| 6.1.6 | PersonLeavesVehicleEvent | 46 |
| 6.2 | BEAM Events | 46 |
| 6.2.1 | ModeChoiceEvent | 46 |
| 6.2.2 | PathTraversalEvent | 47 |
| 7 | Model Inputs | 49 |
| 7.1 | Configuration file | 49 |
| 7.1.1 | Config Options | 50 |
| 8 | Model Outputs | 55 |
| 8.1 | File: /modeChoice.csv | 55 |
| 8.2 | File: /referenceModeChoice.csv | 55 |
| 8.3 | File: /realizedMode.csv | 56 |
| 8.4 | File: /rideHailRevenue.csv | 56 |

| | | |
|-----------|---|-----------|
| 8.5 | File: /ITERS/it.0/0.averageTravelTimes.csv | 56 |
| 8.6 | File: /ITERS/it.0/0.energyUse.png.csv | 56 |
| 8.7 | File: /ITERS/it.0/0.physsimLinkAverageSpeedPercentage.csv | 56 |
| 8.8 | File: /ITERS/it.0/0.physsimFreeFlowSpeedDistribution.csv | 57 |
| 8.9 | File: /ITERS/it.0/0.rideHailWaitingStats.csv | 57 |
| 8.10 | File: /ITERS/it.0/0.rideHailIndividualWaitingTimes.csv | 57 |
| 8.11 | File: /ITERS/it.0/0.rideHailSurgePriceLevel.csv | 57 |
| 8.12 | File: /ITERS/it.0/0.rideHailRevenue.csv | 57 |
| 8.13 | File: /ITERS/it.0/0.tazRideHailSurgePriceLevel.csv | 58 |
| 8.14 | File: /ITERS/it.0/0.rideHailWaitingSingleStats.csv | 58 |
| 8.15 | File: /ITERS/it.0/0.rideHailInitialLocation.csv | 58 |
| 8.16 | File: /stopwatch.txt | 58 |
| 8.17 | File: /scorestats.txt | 59 |
| 8.18 | File: /summaryStats.txt | 59 |
| 8.19 | File: /ITERS/it.0/0.countsCompare.txt | 59 |
| 8.20 | File: /ITERS/it.0/0.events.csv | 60 |
| 8.21 | File: /ITERS/it.0/0.legHistogram.txt | 61 |
| 8.22 | File: /ITERS/it.0/0.rideHailTripDistance.csv | 61 |
| 8.23 | File: /ITERS/it.0/0.tripDuration.txt | 62 |
| 8.24 | File: /ITERS/it.0/0.biasErrorGraphData.txt | 62 |
| 8.25 | File: /ITERS/it.0/0.biasNormalizedErrorGraphData.txt | 62 |
| 9 | Protocols | 63 |
| 9.1 | Trip Planning | 63 |
| 9.1.1 | RoutingRequests | 63 |
| 9.1.2 | ChoosesMode | 64 |
| 9.2 | Traveling | 66 |
| 9.2.1 | Driver | 66 |
| 9.2.2 | Traveler | 68 |
| 9.3 | Household | 71 |
| 9.3.1 | Escort | 71 |
| 9.4 | RideHailing | 71 |
| 9.4.1 | Inquiry | 72 |
| 9.4.2 | Reserve | 73 |
| 9.5 | Transit | 73 |
| 9.6 | Refueling | 74 |
| 9.7 | Modify Passenger Schedule Manager | 74 |
| 10 | DevOps Guide | 75 |
| 10.1 | Git LFS | 75 |
| 10.1.1 | Setup git-lfs Server | 75 |
| 10.2 | Jenkins | 76 |
| 10.2.1 | Setup Jenkins Server | 76 |
| 10.2.2 | Setup Jenkins Slave | 82 |
| 10.2.3 | Configure Jenkins Master | 84 |
| 10.2.4 | Configure Jenkins Jobs | 91 |
| 10.2.5 | Configure Periodic Jobs | 96 |
| 10.2.6 | References | 98 |
| 10.3 | Automated Cloud Deployment | 98 |
| 10.3.1 | Automatic Image (AMI) Update | 98 |
| 11 | Indices and tables | 99 |

BEAM extends the [Multi-Agent Transportation Simulation Framework (MATSim)](<http://www.matsim.org/>) to enable powerful and scalable analysis of urban transportation systems.

Contents:

1.1 Overview

BEAM stands for Behavior, Energy, Autonomy, and Mobility. The model is being developed as a framework for a series of research studies in sustainable transportation at Lawrence Berkeley National Laboratory and the UC Berkeley Institute for Transportation Studies.

BEAM is an extension to the MATSim (Multi-Agent Transportation Simulation) model, where agents employ reinforcement learning across successive simulated days to maximize their personal utility through plan mutation (exploration) and selecting between previously executed plans (exploitation). The BEAM model shifts some of the behavioral emphasis in MATSim from across-day planning to within-day planning, where agents dynamically respond to the state of the system during the mobility simulation. In BEAM, agents can plan across all major modes of travel including driving, walking, biking, transit, and transportation network companies (TNCs).

These key features are summarized here and described in further detail below:

- **MATSim Integration** BEAM leverages the MATSim modeling framework[1], an open source simulation tool with a vibrant community of global developers. MATSim is extensible (BEAM is one of those extensions) which allows modelers to utilize a large suite of tools and plug-ins to serve their research and analytical interests.
- **Resource Markets** While BEAM can be used as a tool for modeling and analyzing the detailed operations of a transportation system, it is designed primarily as an approach to modeling resource markets in the transportation sector. The transportation system is composed of several sets of mobility resources that are in limited supply (e.g. road capacities, vehicle seating, TNC fleet availability, refueling infrastructure). By adopting the MATSim utility maximization approach to achieving user equilibrium for traffic modeling, BEAM is able to find the corresponding equilibrium point across all resource markets of interest.
- **Dynamic Within-Day Planning** Because BEAM places a heavy emphasis on within-day planning, it is possible to simulate modern mobility services in a manner that reflects the emerging transportation system. For example, a virtual TNC in BEAM responds to customer inquiries by reporting the wait time for a ride, which the BEAM agents consider in their decision on what service or mode to use.
- **Rich Modal Choice** BEAM's mode choice model is structured so that agents select modal strategies (e.g. "car" versus "walk to transit" versus "TNC") for each tour prior to the simulation day, but resolve the outcome of these strategies within the day (e.g. route selection, standard TNC versus pooled, etc.). BEAM currently supports a

simple multinomial logit choice model and a more advanced model is under development and will be fully supported by Spring 2018.

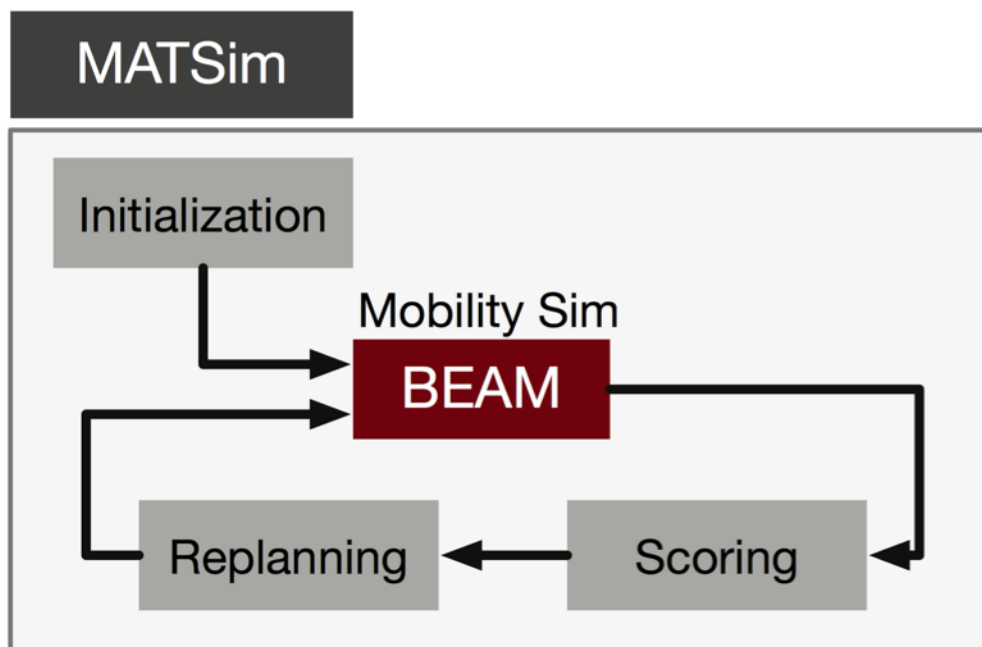
- **Transportation Network Companies** TNCs are already changing the mobility landscape and as driverless vehicles come online, the economics of these services will improve substantially. In BEAM, TNCs are modeled as a fleet of taxis controlled by a centralized manager that responds to requests from customers and dispatches vehicles accordingly. In 2018, BEAM will be extended to simulate the behavioral processes of TNC drivers as well as implement control algorithms designed to optimize fleets of fully automated vehicles.
- **Designed for Scale** BEAM is written primarily in Scala and leverages the [Akka](https://en.wikipedia.org/wiki/Actor_model) library for currency which implements the [Actor Model of Computation](https://en.wikipedia.org/wiki/Actor_model). This approach simplifies the process of deploying transportation simulations at full scale and utilizing high performance computing resources. BEAM has been designed to integrate with Amazon Web Services including a framework to automatically deploy simulation runs to the cloud.

1.2 MATSim Integration

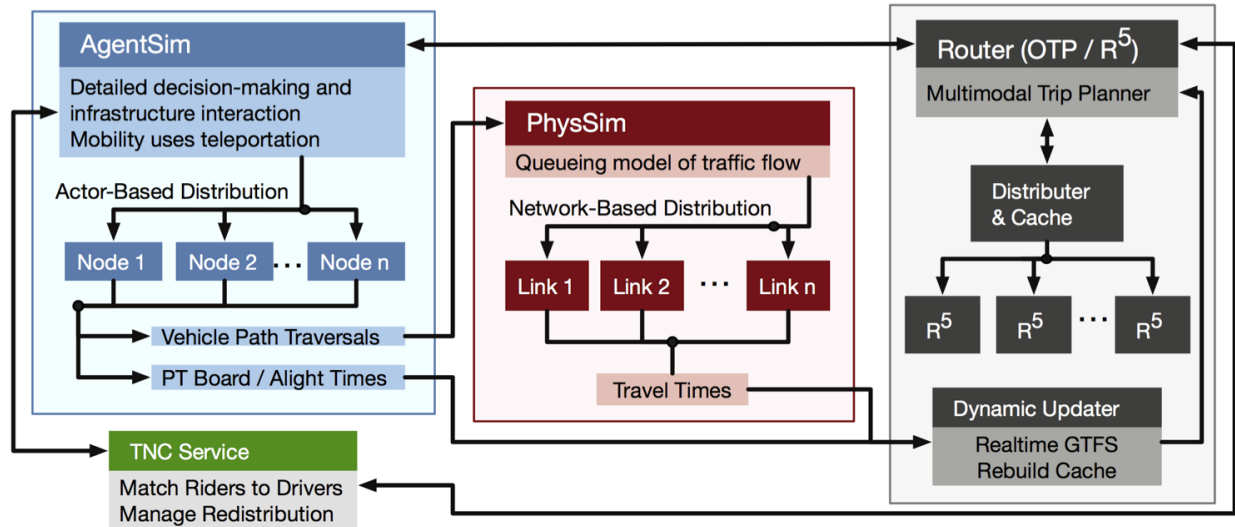
MATSim is a well established agent-based transportation simulation framework with hundreds of users and developers worldwide. BEAM leverages much of the overall structure and conventions of MATSim, but replaces several facilities with new software. The most important of these are the Mobility Simulation and Router.

1.2.1 BeamMobSim

When BEAM is executed, the MATSim engine manages loading of most data (population, households, vehicles, and the network used by PhysSim) as well as executing the MobSim -> Scoring -> Replanning iterative loop. BEAM takes over the MobSim, replacing the legacy MobSim engines (i.e. QSim) with the BeamMobSim.



The BeamMobSim is composed of two simulators, the **AgentSim** and the **PhysSim**. These simulators are related to each other and to the router as illustrated in the following diagram:



1.2.2 AgentSim

The AgentSim is designed to execute the daily plans of the population, allowing the agents to dynamically resolve how limited transportation resources are allocated (see [Resource Markets](#)).

All movements in the AgentSim occur via “teleportation” as discrete events. In other words, given a route and a travel time, an agent simply schedules herself to “arrive” at the destination accordingly. When this occurs, a PathTraversal Event is thrown – one for each vehicle movement, not necessarily each passenger movement – which is used by the PhysSim to simulate traffic flow, resolve congestion, and update travel times in the router so that in future iterations, agents will teleport according to travel times that are consistent with network congestion.

1.2.3 PhysSim

The PhysSim simulates traffic on the road network. The underlying simulation engine is based on the Java Discrete Event Queue Simulator ([JDEQSim](#)) from the MATSim framework. The JDEQSim then simulates traffic flow through the system and updated the Router with new network travel times for use in subsequent iterations.

JDEQSim was designed as a MobSim engine for MATSim, so it is capable of simulating activities and movements through the network. In BEAM, we use JDEQSim within PhysSim as purely a **vehicle** movement simulator. As PathTraversalEvents are received by the PhysSim, a set of MATSim Plans are created, with one plan for each vehicle in the AgentSim. These plans include “Activities” but they are just dummy activities that bracket the movement of each vehicle.

Currently, PhysSim and AgentSim run serially, one after another. This is due to the fact that the PhysSim is substantially faster to run than the AgentSim, because the PhysSim does not need to do any routing calculations. As improvements to AgentSim reduce run times, future versions of BEAM will likely allow AgentSim and PhysSim to run concurrently, or even be run in a tightly coupled manner where each teleportation in AgentSim is replaced with a direct simulation of the propagation of vehicles through the network by the PhysSim.

1.2.4 R5 Router

BEAM uses the [R5 routing engine](#) to accomplish multi-modal routing. Agents from BEAM make request of the router, and the results of the routing calculation are then transformed into objects that are directly usable by the BEAM agents to choose between alternative routes and move throughout the system.

1.2.5 MATSim Events

BEAM adopts the MATSim convention of throwing events that correspond to key moments in the agent's day. But in BEAM, there are two separate event managers, one for the ActorSim and another for the PhysSim.

The standard events output file (e.g. *0.events.csv*) comes from the AgentSim, but in the outputs directory, you will also find an events file from the PhysSim (e.g. *0.physSimEvents.xml.gz*). The events from AgentSim pertain to agents while the events in PhysSim pertain to vehicles. This is an important distinction.

The following MATSim events are thrown within the AgentSim:

- ActivityEndEvent
- PersonDepartureEvent
- PersonEntersVehicleEvent
- PersonLeavesVehicleEvent
- PersonArrivalEvent
- ActivityStartEvent

The following MATSim events are thrown within the PhysSim:

- ActivityEndEvent - these are dummy activities that bracket every vehicle movement
- PersonDepartureEvent - should be interpreted as **vehicle** departure
- LinkEnterEvent
- Wait2LinkEvent / VehicleEntersTraffic
- LinkLeaveEvent
- PersonArrivalEvent - should be interpreted as **vehicle** arrival
- ActivityStartEvent - these are dummy activities that bracket every vehicle movement

Extensions and modules written to observe the above MATSim events can be seamlessly integrated with BEAM in a read-only manner (i.e. for analysis, summary, visualization purposes). However, extensions that are designed to accomplish “within-day” replanning in MATSim will not be directly compatible with BEAM. This is because BEAM already does extensive “within-day” replanning in a manner that is substantially different from QSim.

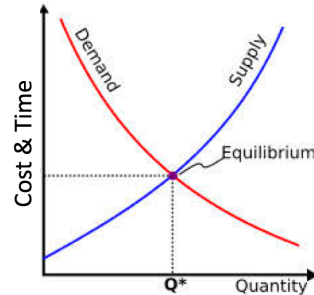
In addition to the standard MATSim events described above, BEAM throws additional events that correspond to the act of choosing a Mode (*ModeChoiceEvent*) and of vehicle movements through the network (*PathTraversalEvent*).

All events (both MATSim and BEAM-specific) and their field descriptions are described in further detail in [Event Specifications](#).

1.3 Resource Markets

BEAM Models Resource Markets

- Resource Markets:
 - Road Capacity
 - Vehicle Capacity
 - TNC Availability
 - Parking
 - Refueling Access



| | | | |
|-------------------|----------|---|--|
| Under Development | Existing | <ul style="list-style-type: none"> • Supply: <ul style="list-style-type: none"> – Driving – Transit (any GTFS) – Walk – TNC (automated) | <ul style="list-style-type: none"> • Demand (governed by behaviors): <ul style="list-style-type: none"> – Mode Choice – Price & Time Sensitive – Route Choice – Multimodal |
| | | <ul style="list-style-type: none"> – Bike – TNC (manual, optimized) – Parking & Refueling | <ul style="list-style-type: none"> – Rerouting – Park Choice – Refuel Choice |

While BEAM can be used as a tool for modeling and analyzing the detailed operations of a transportation system, it is designed primarily as an approach to modeling resource markets in the transportation sector.

The transportation system is composed of several sets of mobility resources that are in limited supply (e.g. road capacities, vehicle seating, TNC fleet availability, refueling infrastructure). With the exception of road capacities, all resources in BEAM are explicitly modeled. For example, there are a finite number of seats available on transit vehicles and there are a finite number of TNC drivers.

As resources are utilized by travelers, they become unavailable to other travelers. This resource competition is resolved dynamically within the AgentSim, making it impossible for multiple agents to simultaneously utilize the same resource.

The degree to which agents use resources is determined both by resource availability and traveler behavior. As the supply of TNC drivers becomes limited, the wait times for hailing a ride increase, which leads to lower utility scores in the mode choice process and therefore reduced consumption of that resource.

By adopting the MATSim utility maximization approach to achieving user equilibrium for traffic modeling, BEAM is able to find the corresponding equilibrium point across all resource markets of interest. Each agent maximizes her utility through the replanning process (which occurs outside the simulation day) as well as within the day through dynamic choice processes (e.g. choosing mode based on with-in day evaluation of modal alternatives).

Ultimately, the combined outcome of running BEAM over successive iterations is a system equilibrium that balances the trade-offs between all resources in the system.

In the figure above, the resource markets that are functioning in BEAM v0.5 are boxed in blue. Future versions of BEAM (planned for 2018) will include the additional resources boxed in red.

1.4 Dynamic Within-Day Planning

Because BEAM places a heavy emphasis on within-day planning, it is possible to simulate modern mobility services in a manner that reflects the emerging transportation system.

For example, a virtual TNC in BEAM responds to customer inquiries by reporting the wait time for a ride, which the BEAM agents consider in their decision on what service or mode to use.

1.5 Rich Modal Choice

BEAM's mode choice model is structured so that agents select modal strategies (e.g. "car" versus "walk to transit" versus "TNC") for each tour prior to the simulation day, but resolve the outcome of these strategies within the day (e.g. route selection, standard TNC versus pooled, etc.). BEAM currently supports a simple multinomial logit choice model and a more advanced model is under development and will be fully supported by Spring 2018.

1.6 Plug-in Electric Vehicle Modeling with BEAM

In 2016, BEAM was originally developed to simulate personally-owned plug-in electric vehicles (PEVs), with an emphasis on detailed representation of charging infrastructure and driver behavior around charging.

In 2017, BEAM underwent a major revision, designed to simulate all modes of travel and to prepare the software for scalability and extensibility. We therefore no longer support the "PEV Only" version of BEAM, though the codebase is still available on the BEAM Github repository under the branch [pev-only](#). In 2018, PEVs will be re-implemented in BEAM following the new framework. In addition, BEAM will support modeling the refueling of fleets of electrified TNCs.

The key features of the "PEV Only" version of BEAM are summarized here and described in further detail in reports linked below.

- **Detailed Representation of Charging Infrastructure** In BEAM, individual chargers are explicitly represented in the region of interest. Chargers are organized as sites that can have multiple charging points which can have multiple plugs of any plug type. The plug types are defined by their technical characteristics (i.e. power capacity, price, etc.) and their compatibility with vehicles types (e.g. Tesla chargers vs. CHAdeMO vs. SAE). Physical access to chargers is also represented explicitly, i.e., charging points can only be accessed by a limited number of parking spaces. Chargers are modeled as queues, which can be served in an automated fashion (vehicle B begins charging as soon as vehicle A ends) or manually by sending notifications to agents that it is their turn to begin a charging session.
- **Robust Behavioral Modeling** The operational decisions made by PEV drivers are modeled using discrete choice models, which can be parameterized based on the outcomes of stated preference surveys or revealed preference analyses. For example, the decision of whether and where to charge is currently modeled in BEAM as a nested logit choice that considers a variety of factors including the location, capacity, and price of all chargers within a search radius in addition to the state of charge of the PEV and features of the agent's future mobility needs for the day. The utility functions for the model are in part based on empirical work by Wen et al.[2] who surveyed PEV drivers and analyzed the factors that influence their charging decisions.

1.7 Contact Information

Primary Technical Contacts:

Colin Sheppard colin.sheppard@lbl.gov

Rashid Waraich rwaraich@lbl.gov

1.8 Reports and Papers

“Modeling Plug-in Electric Vehicle Trips, Charging Demand and Infrastructure”.

1.9 References

1. Horni, A., Nagel, K. and Axhausen, K.W. (eds.) 2016 *The Multi-Agent Transport Simulation MATSim*. London: Ubiquity Press. DOI: <http://dx.doi.org/10.5334/baw>. License: CC-BY 4.0.
2. Wen, Y., MacKenzie, D. & Keith, D. Modeling the Charging Choices of Battery Electric Vehicle Drivers Using Stated Preference Data. TRB Proc. Pap. No 16-5618

2.1 Getting Started

The following guide is designed as a demonstration of using BEAM and involves running the model on a scaled population and transportation system. This is the ideal place to familiarize yourself with the basics of configuring and running BEAM as well as doing small scale tests and analysis.

For more advanced utilization or to contribute to the BEAM project, see the *Developer's Guide*.

2.1.1 System Requirements

- At least 8GB RAM
- Windows, Mac OSX, Linux
- Java Runtime Environment or Java Development Kit 1.8
- To verify your JRE: https://www.java.com/en/download/help/version_manual.xml
- To download JRE 1.8 (AKA JRE 8): <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>
- We also recommend downloading the VIA visualization app and obtaining a free or paid license: <https://simunto.com/via/>
- Git and Git-LFS

2.1.2 Prerequisites :

Install Java

BEAM requires Java 1.8 JDK / JRE to be installed. If a different version of java is already installed on your system, please upgrade the version to 1.8. See this [link](#) for steps to check the current version of your JRE.

If java is not already installed on your system , please follow this [download manual](#) to install java on your system.

Please note that BEAM is currently compatible only with Java 1.8 and is not compatible with any of the older or recent versions.

Install Gradle

BEAM uses [gradle](#) as its build tool. If gradle is not already installed, check this [gradle installation guide](#) for steps on how to download and install gradle. Once gradle is successfully installed, verify the installation by running the command

```
gradle
```

2.1.3 GIT-LFS Configuration

The installation process for git-lfs client(v2.3.4, latest installer has some issue with node-git-lfs) is very simple. For detailed documentation please consult [github guide](#) for Mac, windows and Linux.

To verify successful installation, run following command:

```
$ git lfs install
Git LFS initialized.
```

To confirm that you have installed the correct version of client run the following command:

```
$ git lfs env
```

It will print out the installed version, and please make sure it is *git-lfs/2.3.4*.

To update the text pointers with the actual contents of files, run the following command (if it requests credentials, use any username and leave the password empty):

```
$ git lfs pull
Git LFS: (98 of 123 files) 343.22 MB / 542.18 MB
```

Installing git lfs on windows :

With Git LFS windows installation, it is common to have the wrong version of git-lfs installed, because in these latest git client version on windows, git lfs comes packaged with the git client.

When installing the git client one needs to uncheck git lfs installation. If mistakenly you installed git lfs with the git client, the following steps are needed to correct it (uninstalling git lfs and installing the required version does not work...):

- Uninstall git
- Install the git client (uncheck lfs installation)
- Install git lfs version 2.3.4 separately

Another alternative to above is to get the old git-lfs.exe from the release archives and replace the executable found in *[INSTALL PATH]mingw64\bin* and *[INSTALL PATH]cmd*, where the default installation path usually is *C:\Program Files\Git*

2.1.4 Installing BEAM

Clone the beam repository:

```
git clone git@github.com:LBLN-UCB-STI/beam.git
```

Change directories into that repository:

```
cd beam
```

Now checkout the latest stable version of BEAM, v0.7.0:

```
git checkout v0.7.0
```

Run the gradle command to compile BEAM, this will also download all required dependencies automatically:

```
gradle classes
```

Now you're ready to run BEAM!

2.1.5 Running BEAM

Inside of the repository is a folder 'test/input' containing several scenarios and configurations you can experiment with.

The simplest, smallest, and fastest is the beamville scenario (described below). Try to run beamville with this command:

```
./gradlew :run -PappArgs=["--config", 'test/input/beamville/beam.conf']"
```

The BEAM application by default sets max RAM allocation to 140g (see **maxRAM** setting in gradle.properties). This needs to be adjusted based on the available memory on your system.

The max allocatable RAM value can be overridden by setting the environment variable **MAXRAM** to the required value.

On Ubuntu , the environment variable can be set using the below command :

```
export MAXRAM=10g
```

where 10g = 10GB

Similarly on windows it can be set using the below command :

```
setx MAXRAM="10g"
```

The outputs are written to the 'output' directory, should see results appear in a sub-folder called "beamville_%DATE_TIME%".

Optionally you can also run BEAM from your favourite IDE . Check the below section on how to configure and run BEAM using IntelliJ IDEA.

2.1.6 Running BEAM with IntelliJ IDE

IntelliJ IDEA community edition is an open source IDE available for free. It can be downloaded from [here](#)

After successful download , run the executable and follow the installation wizard to install IntelliJ IDEA.

When running the IDE for the first time , it asks to import previous settings (if any) from a local path, if no previous settings to choose , select "Do not import settings" and click Ok.

Importing BEAM project into IDE

Once the IDE is successfully installed , proceed with the below steps to import BEAM into the IDE.

1. **Open the IDE and agree to the privacy policy and continue**

(Optional) IDEA walks you through some default configurations set up here . In case you want to skip these steps , click

- Select a UI theme of choice and go to Next: Default Plugins
- Select only the required plugins (gradle , java are mandatory) and disable the others and go to Next:Feature plugins
- Install scala and click “Start using IntelliJ IDEA”

2. In the welcome menu , select “Import Project” and provide the location of the locally cloned BEAM project

3. Inside the import project screen, select “Import project from external model” and choose “Gradle” from the available and click Next

4. Click Finish.

The project should now be successfully imported into the IDE and a build should be initiated automatically. If no build is triggered automatically , you can manually trigger one by going to Build > Build Project.

Installing scala plugin

If optional configuration in step 1 of above section was skipped , scala plugin will not be added automatically . To manually enable scala plugin go to File > Settings > Plugins. Search for scala plugin and click Install.

Setting up scala SDK

Since BEAM is built with java/scala . A scala sdk module needs to be configured to run BEAM. Check the below steps on how to add a scala module to IDEA * Go to File > Project Settings > Global Libraries * Click + and select Scala SDK * Select the required scala SDK from the list , if no SDK found click Create. * Click “Browse” and select the scala home path or click “Download” (choose 2.12.x version)

Running BEAM from IDE

BEAM requires some arguments to be specified during run-time like the scenario configuration. These configuration settings can be added as a run configuration inside the IDE.

Steps to add a new configuration :

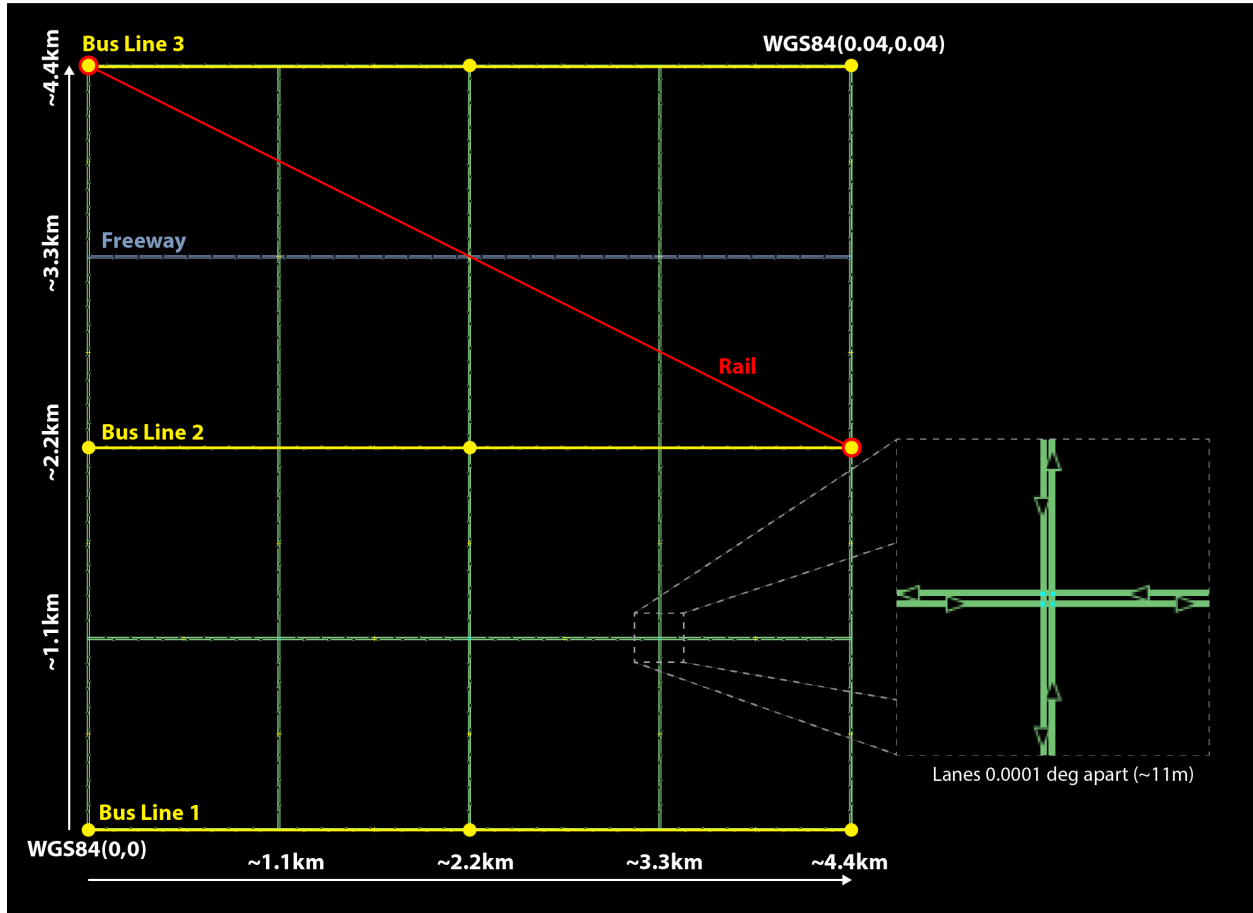
- Go to Run > Edit Configurations
- Click + and from the templates list and select “Application”
- Fill in the following values
 - Main Class : beam.sim.RunBeam
 - VM options : -Xmx8g
 - Program Arguments : -config test/input/beamville/beam.conf (this runs beamville scenario, changes the folder path to run a different scenario)
 - Working Directory : /home/beam/BEAM
 - Environment Variables : PWD=/home/beam/BEAM
 - use submodule of path : beam.beam.main
- Click Ok to save the configuration.

To add a configuration for a different scenario , follow the above steps and change the folder path to point to the required scenario in program arguments

2.1.7 Scenarios

We have provided two scenarios for you to explore under the *test/input* directory.

The *beamville* test scenario is a toy network consisting of a 4 x 4 block gridded road network, a light rail transit agency, a bus transit agency, and a population of ~50 agents.



The *sf-light* scenario is based on the City of San Francisco, including the SF Muni public transit service and a range of sample populations from 1000 to 25,000 agents.



2.1.8 Inputs

For more detailed inputs documentation, see [Model Inputs](#).



































BEAM follows the [MATSim convention](#) for most of the inputs required to run a simulation, though some inputs files can alternatively be provided in CSV instead of XML format. Also, the road network and transit system inputs are based on the [R5 requirements](#). The following is a brief overview of the minimum requirements needed to conduct a BEAM run.

- A configuration file (e.g. *beam.conf*)
- The person population and corresponding attributes files (e.g. *population.xml* and *populationAttributes.xml*)
- The household population and corresponding attributes files (e.g. *households.xml* and *householdAttributes.xml*)
- The personal vehicle fleet (e.g. *vehicles.csv*)
- The definition of vehicle types including for personal vehicles and the public transit fleet (e.g. *vehicleTypes.csv*)

- A directory containing network and transit data used by R5 (e.g. *r5/*)
- The open street map network (e.g. *r5/beamville.osm*)
- GTFS archives, one for each transit agency (e.g. *r5/bus.zip*)

2.1.9 Outputs

At the conclusion of a BEAM run using the default *beamville* scenario, the output files in the should look like this when the run is complete:

| Name | |
|---|--|
| ▼ | ITERS |
| ▼ | it.0 |
|  | 0.average_travel_times_car.png |
|  | 0.average_travel_times_drive_transit.png |
|  | 0.average_travel_times_ride_hailing.png |
|  | 0.average_travel_times_walk_transit.png |
|  | 0.average_travel_times_walk.png |
|  | 0.energy_use.png |
|  | 0.events.csv |
|  | 0.expectedMaxUtilityHeatMap.csv |
|  | 0.legHistogram_all.png |
|  | 0.legHistogram_car.png |
|  | 0.legHistogram_drive_transit.png |
|  | 0.legHistogram_ride_hailing.png |
|  | 0.legHistogram_walk_transit.png |
|  | 0.legHistogram_walk.png |
|  | 0.legHistogram.txt |
|  | 0.mode_choice.png |
|  | 0.passenger_per_trip_bus.png |
|  | 0.passenger_per_trip_car.png |
|  | 0.passenger_per_trip_subway.png |
|  | 0.physsim-plans.xml.gz |
|  | 0.physSimEvents.xml.gz |
|  | 0.plans.xml.gz |
|  | 0.tnc_deadheading_distance.png |
|  | 0.tnc_passenger_per_trip.png |
|  | 0.tripdurations.txt |
| ▼ | it.1 |
|  | 1.average_travel_times_car.png |
|  | 1.average_travel_times_drive_transit.png |
|  | 1.average_travel_times_ride_hailing.png |
|  | 1.average_travel_times_walk_transit.png |
|  | 1.average_travel_times_walk.png |
|  | 1.energy_use.png |
|  | 1.events.csv |
|  | 1.legHistogram_all.png |
|  | 1.legHistogram_car.png |

Each iteration of the run produces a sub-folder under the *ITERS* directory. Within these, several automatically generated outputs are written including plots of modal usage, TNC dead heading, and energy consumption by mode.

In addition, raw outputs are available in the two events file (one from the AgentSim and one from the PhysSim, see *MATSim Events* for more details), titled *%ITER%.events.csv* and *%ITER%.physSimEvents.xml.gz* respectively.

2.1.10 Model Config

To get started, we will focus your attention on a few of the most commonly used and useful configuration parameters that control beam:

```
# Ride Hailing Params
beam.agentsim.agents.rideHail.initialization.procedural.
↪numDriversAsFractionOfPopulation=0.05
beam.agentsim.agents.rideHail.defaultCostPerMile=1.25
beam.agentsim.agents.rideHail.defaultCostPerMinute=0.75
# Scaling and Tuning Params; 1.0 results in no scaling
beam.agentsim.tuning.transitCapacity = 0.2
beam.agentsim.tuning.transitPrice = 1.0
beam.agentsim.tuning.tollPrice = 1.0
beam.agentsim.tuning.rideHailPrice = 1.0
```

- `numDriversAsFractionOfPopulation` - Defines the # of ride hailing drivers to create. Drivers begin the simulation located at or near the homes of existing agents, uniformly distributed.
- `defaultCostPerMile` - One component of the 2 part price of ride hail calculation.
- `defaultCostPerMinute` - One component of the 2 part price of ride hail calculation.
- `transitCapacity` - Scale the number of seats per transit vehicle... actual seats are rounded to nearest whole number. Applies uniformly to all transit vehicles.
- `transitPrice` - Scale the price of riding on transit. Applies uniformly to all transit trips.
- `tollPrice` - Scale the price to cross tolls.
- `rideHailPrice` - Scale the price of ride hailing. Applies uniformly to all trips and is independent of `defaultCostPerMile` and `defaultCostPerMinute` described above. I.e. $price = (costPerMile + costPerMinute) * rideHailPrice$

2.2 Experiment Manager

BEAM features a flexible experiment manager which allows users to conduct multi-factorial experiments with minimal configuration. The tool is powered by Jinja templates (see more <http://jinja.pocoo.org/docs/2.10/>).

We have created two example experiments to demonstrate how to use the experiment manager. The first is a simple 2-factorial experiment that varies some parameters of scientific interest. The second involves varying parameters of the mode choice model as one might do in a calibration exercise.

In any experiment, we seek to vary the parameters of BEAM systematically and producing results in an organized, predictable location to facilitate post-processing. For the two factor experiment example, we only need to vary the contents of the BEAM config file (`beam.conf`) in order to achieve the desired analysis.

Lets start from building your experiment definitions in `experiment.yml` (see example in `test/input/beamville/example-experiment/experiment.yml`). `experiment.yml` is a YAML config file which consists of 3 sections: header, default-Params, and factors.

The Header defines the basic properties of the experiment, the title, author, and a path to the configuration file (paths should be relative to the project root):

```
title: Example-Experiment
author: MyName
beamTemplateConfPath: test/input/beamville/beam.conf
```

The Default Params are used to override any parameters from the BEAM config file for the whole experiment. These values can, in turn, be overridden by factor levels if specified. This section is mostly a convenient way to ensure certain parameters take on specific values without modifying the BEAM config file in use.

Experiments consist of ‘factors’, which are a dimension along which you want to vary parameters. Each instance of the factor is a level. In our example, one factor is “transitCapacity” consisting of two levels, “Low” and “High”. You can think about factors as of main influencers (or features) of simulation model while levels are discrete values of each factor.

Factors can be designed however you choose, including adding as many factors or levels within those factors as you want. E.g. to create a 3 x 3 experimental design, you would set three levels per factor as in the example below:

```
factors:
- title: transitCapacity
  levels:
  - name: Low
    params:
      beam.agentsim.tuning.transitCapacity: 0.01
  - name: Base
    params:
      beam.agentsim.tuning.transitCapacity: 0.05
  - name: High
    params:
      beam.agentsim.tuning.transitCapacity: 0.1
- title: ridehailNumber
  levels:
  - name: Low
    params:
      beam.agentsim.agents.rideHail.numDriversAsFractionOfPopulation: 0.001
  - name: Base
    params:
      beam.agentsim.agents.rideHail.numDriversAsFractionOfPopulation: 0.01
  - name: High
    params:
      beam.agentsim.agents.rideHail.numDriversAsFractionOfPopulation: 0.1
```

Each level and the baseScenario defines *params*, or a set of key,value pairs. Those keys are either property names from beam.conf or placeholders from any template config files (see below for an example of this). Param names across factors and template files must be unique, otherwise they will overwrite each other.

In our second example (see directory *test/input/beamville/example-calibration/*), we have added a template file *modeChoiceParameters.xml.tpl* that allows us to change the values of parameters in BEAM input file *modeChoiceParameters.xml*. In the *experiment.yml* file, we have defined 3 factors with two levels each. One level contains the property *mnl_ride_hail_intercept*, which appears in *modeChoiceParameters.xml.tpl* as `{{ mnl_ride_hail_intercept }}`. This placeholder will be replaced during template processing. The same is true for all properties in the defaultParams and under the facts. Placeholders for template files must NOT contain the dot symbol due to special behaviour of Jinja. However it is possible to use the full names of properties from *beam.conf* (which *do* include dots) if they need to be overridden within this experiment run.

Also note that *mnl_ride_hail_intercept* appears both in the level specification and in the baseScenario. When using a template file (versus a BEAM Config file), each level can only override properties from Default Params section of *experiment.yml*.

Experiment generation can be run using following command:

```
gradle -PmainClass=beam.experiment.ExperimentGenerator -PappArgs="['--experiments',
↪ 'test/input/beamville/example-experiment/experiment.yml']" execute
```

It's better to create a new sub-folder (e.g. 'calibration' or 'experiment-1') in your data input directory and put both templates and the experiment.yml there. The ExperimentGenerator will create a sub-folder next to experiment.yml named *runs* which will include all of the data needed to run the experiment along with a shell script to execute a local run. The generator also creates an *experiments.csv* file next to experiment.yml with a mapping between experimental group name, the level name and the value of the params associated with each level.

Within each run sub-folder you will find the generated BEAM config file (based on beamTemplateConfPath), any files from the template engine (e.g. *modeChoiceParameters.xml*) with all placeholders properly substituted, and a *runBeam.sh* executable which can be used to execute an individual simulation. The outputs of each simulation will appear in the *output* subfolder next to runBeam.sh

2.3 Calibration

This section describes calibrating BEAM simulation outputs to achieve real-world targets (e.g., volumetric traffic counts, mode splits, transit boarding/alighting, etc.). A large number of parameters affect simulation behavior in complex ways such that grid-search tuning methods would be extremely time-consuming. Instead, BEAM uses SigOpt, which uses Bayesian optimization to rapidly tune scenarios as well as analyze the sensitivity of target metrics to parameters.

2.3.1 Optimization-based Calibration Principles

At a high level, the SigOpt service seeks to find the *optimal value*, p^* of an *objective*, $f_0 : \mathbb{R}^n \rightarrow \mathbb{R}$, which is a function of a vector of *decision variables* $x \in \mathbb{R}^n$ subject to *constraints*, $f_i : \mathbb{R}^n \rightarrow \mathbb{R}, i = 1, \dots, m$.

In our calibration problem, p^* represents the value of a *metric* representing an aggregate measure of some deviation of simulated values from real-world values. Decision variables are hyperparameters defined in the *.conf* file used to configure a BEAM simulation. The constraints in this problem are the bounds within which it is believed that the SigOpt optimization algorithm should search. The calibration problem is solved by selecting values of the hyperparameters that minimize the output of the objective function.

Operationally, for each calibration attempt, BEAM creates an *Experiment* using specified *Parameter* variables, their *Bounds*'s, and the number of workers (applicable only when using parallel calibration execution) using the SigOpt API. The experiment is assigned a unique ID and then receives a *Suggestion* (parameter values to simulate) from the SigOpt API, which assigns a value for each *Parameter*. Once the simulation has completed, the metric (an implementation of the *beam.calibration.api.ObjectiveFunction* interface) is evaluated, providing an *Observation* to the SigOpt API. This completes one iteration of the calibration cycle. At the start of the next iteration new *Suggestion* is returned by SigOpt and the simulation is re-run with the new parameter values. This process continues for the number of iterations specified in a command-line argument.

Note: that this is a different type of iteration from the number of iterations of a run of BEAM itself. Users may wish to run BEAM for several iterations of the co-evolutionary plan modification loop prior to evaluating the metric.

2.3.2 SigOpt Setup

Complete the following steps in order to prepare your simulation scenarios for calibration with SigOpt:

1. [Sign up](#) for a SigOpt account (note that students and academic researchers may be able to take advantage of educational pricing options).

2. Log-in to the SigOpt web interface.
3. Under the [API Tokens](#) menu, retrieve the **API Token** and **Development Token** add the tokens as environmental variables in your execution environment with the keys `SIGOPT_API_TOKEN` and `SIGOPT_DEV_API_TOKEN`.

2.3.3 Configuration

Prepare YAML File

Configuring a BEAM scenario for calibration proceeds in much the same way as it does for an experiment using the *Experiment Manager*. In fact, with some minor adjustments, the *YAML* text file used to define experiments has the same general structure as the one used to specify tuning hyperparameters and ranges for calibration (see example file `beam/test/input/beamville/example-calibration/experiment.yml`):

```
title: this is the name of the SigOpt experiment
beamTemplateConfPath: the config file to be used for the experiments
modeChoiceTemplate: mode choice template file
numWorkers: this defines for a remote run, how many parallel runs should be executed,
↳ (number of machines to be started)
params:
  ### ---- run template env variables ---###
  EXPERIMENT_MAX_RAM: 16g (might be removed in future)
  S3_OUTPUT_PATH_SUFFIX: "sf-light" (might be removed in future)
  DROP_OUTPUT_ONCOMPLETE: "true" (might be removed in future)
  IS_PARALLEL: "false" (might be removed in future)

runName: instance name for remote run
beamBranch: branch name
beamCommit: commit hash
deployMode: "execute"
executeClass: "beam.calibration.RunCalibration"
beamBatch: "false"
shutdownWait: "15"
shutdownBehavior: "stop"
s3Backup: "true"
maxRAM: "140g"
region: "us-west-2"
instanceType: "m4.16xlarge"
```

The major exceptions are the following:

- Factors may have only a single numeric parameter, which may (at the moment) only take two levels (High and Low).

These act as bounds on the values that SigOpt will try for a particular decision variable.

- The level of parallelism is controlled by a new parameter in the header called *numberOfWorkers*. Setting its value

above 1 permits running calibrations in parallel in response to multiple concurrent open *Suggestions*.

Create Experiment

Use `beam.calibration.utils.CreateExperiment` to create a new SigOpt experiment. Two inputs are needed for this: a *YAML* file and a *benchmark.csv* file (this second parameter might be removed in the near future, as not needed).

After running the script you should be able to see the newly created experiment in the SigOpt web interface and the experiment id is also printed out in the console.

Set in Config

One must also select the appropriate implementation of the *ObjectiveFunction* interface in the *.conf* file pointed to in the *YAML*, which implicitly defines the metric and input files. Several example implementations are provided such as *ModeChoiceObjectiveFunction*. This implementation compares modes used at the output of the simulation with benchmark values. To optimize this objective, it is necessary to have a set of comparison benchmark values, which are placed in the same directory as other calibration files:

```
beam.calibration.objectiveFunction = "ModeChoiceObjectiveFunction_
↳ AbsolutErrorWithPreferenceForModeDiversity"
beam.calibration.mode.benchmarkFileLoc=${beam.inputDirectory}"/calibration/benchmark.
↳ csv"
```

(Needed for scoring funtions which try to match mode share).

2.3.4 Execution

Execution of a calibration experiment requires running the *beam.calibration.RunCalibration* class using the following arguments:

| | |
|------------------------|--|
| --experiments | production/application-sfbay/calibration/experiment_counts_calibration.yml |
| --benchmark | Location of the benchmark file (production/applicaion-sfbay/calibration/benchmark.csv) |
| --num_iters | Number of SigOpt iterations to be conducted (in series). |
| --experiment_id | If an <i>experimentID</i> has already been defined, add it here to continue an experiment or put |

“None” to start a new experiment.

| | |
|-------------------|------------------------|
| --run_type | Can be local or remote |
|-------------------|------------------------|

2.3.5 Manage Experiment

As the number of open suggestions for an experiment is limited (10 in our case), we sometimes might need to cleanup suggestions mauully using *beam.calibration.utils.DeleteSuggestion* script to both delete specific and all open suggestions (e.g. if there was an exception during all runs and need to restart).

2.4 Timezones and GTFS

There is a subtle requirement in BEAM related to timezones that is easy to miss and cause problems.

BEAM uses the R5 router, which was designed as a stand-alone service either for doing accessibility analysis or as a point to point trip planner. R5 was designed with public transit at the top of the developers’ minds, so they infer the time zone of the region being modeled from the “timezone” field in the “agency.txt” file in the first GTFS data archive that is parsed during the network building process.

Therefore, if no GTFS data is provided to R5, it cannot infer the locate timezone and it then assumes UTC.

Meanwhile, there is a parameter in beam, “beam.routing.baseDate” that is used to ensure that routing requests to R5 are send with the appropriate timestamp. This allows you to run BEAM using any sub-schedule in your GTFS archive. I.e. if your base date is a weekday, R5 will use the weekday schedules for transit, if it’s a weekend day, then the weekend schedules will be used.

The time zone in the `baseDate` parameter (e.g. for PST one might use “2016-10-17T00:00:00-07:00”) must match the time zone in the GTFS archive(s) provided to R5.

As a default, we provide a “dummy” GTFS data archive that is literally empty of any transit schedules, but is still a valid GTFS archive. This archive happens to have a time zone of Los Angeles. You can download a copy of this archive here:

<https://www.dropbox.com/s/2tfbhxuvmep7wf7/dummy.zip?dl=1>

But in general, if you use your own GTFS data for your region, then you may need to change this `baseDate` parameter to reflect the local time zone there. Look for the “`timezone`” field in the “`agency.txt`” data file in the GTFS archive.

The date specified by the `baseDate` parameter must fall within the schedule of all GTFS archives included in the R5 sub-directory. See the “`calendar.txt`” data file in the GTFS archive and make sure your `baseDate` is within the “`start_date`” and “`end_date`” fields folder across all GTFS inputs. If this is not the case, you can either change `baseDate` or you can change the GTFS data, expanding the date ranges... the particular dates chosen are arbitrary and will have no other impact on the simulation results.

One more word of caution. If you make changes to GTFS data, then make sure you properly zip the data back into an archive. You do this by selecting all of the individual text files and then right-click-compress. Do not compress the folder containing the GTFS files, if you do this, R5 will fail to read your data and will do so without any warning or errors.

Finally, any time you make a changes to either the GTFS inputs or the OSM network inputs, then you need to delete the file “`network.dat`” under the “`r5`” sub-directory. This will signal to the R5 library to re-build the network.

2.5 Converting a MATSim Scenario to Run with BEAM

The following MATSim input data are required to complete the conversion process:

- Matsim network file: (e.g. `network.xml`)
- Matsim plans (or population) file: (e.g. `population.xml`)
- A download of OpenStreetMap data for a region that includes your region of interest. Should be in pbformat. For North American downloads: <http://download.geofabrik.de/north-america.html>

The following inputs are optional and only recommended if your MATSim scenario has a constrained vehicle stock (i.e. not every person owns a vehicle):

- Matsim vehicle definition (e.g. `vehicles.xml`)
- Travel Analysis Zone shapefile for the region, (e.g. as can be downloaded from https://www.census.gov/geo/maps-data/data/cbf/cbf_taz.html)

Finally, this conversion can only be done with a clone of the full BEAM repository. Gradle commands will **not** work with releases: <https://github.com/LBNL-UCB-STI/beam/releases>

2.5.1 Conversion Instructions

Note that we use the MATSim Sioux Falls scenario as an example. The data for this scenario are already in the BEAM repository under “`test/input/siouxfalls`”. We recommend that you follow the steps in this guide with that data to produce a working BEAM Sioux Falls scenario and then attempt to do the process with your own data.

1. Create a folder for your scenario in project directory under `test/input` (e.g: `test/input/siouxfalls`)
2. Create a sub-directory to your scenario directory and name it “`conversion-input`” (exact name required)
3. Create a another sub-directory and name it “`r5`”.

4. Copy the MATSim input data to the conversion-input directory.
5. Copy the BEAM config file from test/input/beamville/beam.conf into the scenario directory and rename to your scenario (e.g. test/input/siouxfalls/siouxfalls.conf)
6. Make the following edits to siouxfalls.conf (or your scenario name, replace Sioux Falls names below with appropriate names from your case):
 - Do a global search/replace and search for “beamville” and replace with your scenario name (e.g. “siouxfalls”).
 - matsim.conversion.scenarioDirectory = “test/input/siouxfalls”
 - matsim.conversion.populationFile = “Siouxfalls_population.xml” (just the file name, assumed to be under conversion-input)
 - matsim.conversion.matsimNetworkFile = “Siouxfalls_network_PT.xml” (just the file name, assumed to be under conversion-input)
 - matsim.conversion.generateVehicles = true (If true – common – the conversion will use the population data to generate default vehicles, one per agent)
 - matsim.conversion.vehiclesFile = “Siouxfalls_vehicles.xml” (optional, if generateVehicles is false, specify the matsim vehicles file name, assumed to be under conversion-input)
 - matsim.conversion.defaultHouseholdIncome (an integer to be used for default household incomes of all agents)
 - matsim.conversion.osmFile = “south-dakota-latest.osm.pbf” (the Open Street Map source data file that should be clipped to the scenario network, assumed to be under conversion-input)
 - matsim.conversion.shapeConfig.shapeFile (file name shape file package, e.g: for shape file name tz46_d00, there should be following files: tz46_d00.shp, tz46_d00.dbf, tz46_d00.shx)
 - matsim.conversion.shapeConfig.tazIdFieldName (e.g. “TZ46_D00_I”, the field name of the TAZ ID in the shape file)
 - beam.spatial.localCRS = “epsg:26914” (the local EPSG CRS used for distance calculations, should be in units of meters and should be the CRS used in the network, population and shape files)
 - beam.routing.r5.mNetBuilder.toCRS = “epsg:26914” (same as above)
 - beam.spatial.boundingBoxBuffer = 10000 (meters to pad bounding box around the MATSim network when clipping the OSM network)
 - The BEAM parameter beam.routing.baseDate has a time zone (e.g. for PST one might use “2016-10-17T00:00:00-07:00”). This time zone must match the time zone in the GTFS data provided to the R5 router. As a default, we provide the latest GTFS data from the City of Sioux Falls (“siouxareametro-sd-us.zip”. downloaded from transitland.org) with a timezone of America/Central. But in general, if you use your own GTFS data for your region, then you may need to change this baseDate parameter to reflect the local time zone there. Look for the “timezone” field in the “agency.txt” data file in the GTFS archive. Finally, the date specified by the baseDate parameter must fall within the schedule of all GTFS archives included in the R5 sub-directory. See the “calendar.txt” data file in the GTFS archive and make sure your baseDate is within the “start_date” and “end_date” fields folder across all GTFS inputs. If this is not the case, you can either change baseDate or you can change the GTFS data, expanding the date ranges. ... the particular dates chosen are arbitrary and will have no other impact on the simulation results.
8. Run the conversion tool
 - Open command line in beam root directory and run the following command, replace [/path/to/conf/file] with the path to your config file: `gradlew matsimConversion -PconfPath=[/path/to/conf/file]`

The tool should produce the following outputs:

- householdAttributes.xml

- households.xml
- population.xml
- populationAttributes.xml
- taz-centers.csv
- transitVehicles.xml
- vehicles.xml

9. Run OSMOSIS

The console output should contain a command for the osmosis tool, a command line utility that allows you manipulate OSM data. If you don't have osmosis installed, download and install from: <https://wiki.openstreetmap.org/wiki/Osmosis>

Copy the osmosis command generated by conversion tool and run from the command line from within the BEAM project directory:

```
osmosis      -read-pbf      file=/path/to/osm/file/south-dakota-latest.osm.pbf      -bounding-box
top=43.61080226522504      left=-96.78138443934351      bottom=43.51447260628691      right=-
96.6915507011093 completeWays=yes completeRelations=yes clipIncompleteEntities=true -write-pbf
file=/path/to/dest-osm.pbf
```

10. Run BEAM

- Main class to execute: beam.sim.RunBeam
- VM Options: -Xmx2g (or more if a large scenario)
- Program arguments, path to beam config file from above, (e.g. -config "test/input/siouxfalls/siouxfalls.conf")
- Environment variables: PWD=/path/to/beam/folder

3.1 Repositories

The beam repository on github is [here](#).

The convention for merging into the master branch is that master needs to be pass all tests and at least one other active BEAM developer needs to review your changes before merging. Please do this by creating a pull request from any new feature branches into master. We also encourage you to create pull requests early in your development cycle which gives other's an opportunity to observe and/or provide feedback in real time. When you are ready for a review, invite one or more through the pull request.

Please use the following naming convention for feature branches, "<initials-or-username>/<descriptive-feature-branch-name>". Adding the issue number is also helpful, e.g.:

cjrs/issue112-update-docs

An example workflow for contributing a new feature beam might look like this:

- create a new branch off of master (e.g. cjrs/issue112-update-docs)
- push and create a pull request right away
- work in cjrs/issue112-update-docs
- get it to compile, pass tests
- request reviews from pull request
- after reviews and any subsequent iterations, merge into master and close pull request
- delete feature branch unless continued work to happy imminently on same feature branch

The pev-only and related feature branches hold a previous version of BEAM (v0.1.X) which is incompatible with master but is still used for modeling and analysis work.

3.2 Configuration

We use `typesafe config` for our configuration parser and we use the `tscfg` utility for generating typesafe container class for our config that we can browse with auto-complete while developing.

Then you can make a copy of the config template under:

```
src/main/resources/config-template.conf
```

and start customizing the configurations to your use case.

To add new parameters or change the structure of the configuration class itself, simply edit the `config-template.conf` file and run the gradle task:

```
gradle generateConfig
```

This will generate a new class `src/main/scala/beam/metasim/config/BeamConfig.scala` which will reflect the new structure and parameters.

3.3 Environment Variables

BEAM supports using an environment variable to optionally specify a directory to write outputs. This is not required.

Depending on your operating system, the manner in which you want to run the BEAM application or gradle tasks, the specific place where you set these variables will differ. To run from the command line, add these statements to your `.bash_profile` file:

```
export BEAM_OUTPUT=/path/to/your/preferred/output/destination/`
```

To run from IntelliJ as an “Application”, edit the “Environment Variables” field in your Run Configuration to look like this:

```
BEAM_OUTPUT="/path/to/your/preferred/output/destination/"
```

Finally, if you want to run the gradle tasks from IntelliJ in OS X, you need to configure your variables as launch tasks by creating a plist file for each. The files should be located under `~/Library/LaunchAgents/` and look like the following. Note that after creating the files you need to log out / log in to OS X and you can’t Launch IntelliJ automatically on log-in because the LaunchAgents might not complete in time.

File: `~/Library/LaunchAgents/setenv.BEAM_OUTPUT.plist`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Label</key>
    <string>setenv.BEAM_OUTPUT</string>
    <key>ProgramArguments</key>
    <array>
      <string>/bin/launchctl</string>
      <string>setenv</string>
      <string>BEAM_OUTPUT</string>
      <string>/path/to/your/preferred/output/destination/</string>
    </array>
  </dict>
</plist>
```

(continues on next page)

(continued from previous page)

```
<key>RunAtLoad</key>
<true/>
<key>ServiceIPC</key>
<false/>
</dict>
</plist>
```

3.4 GIT-LFS timeout - how to proceed

Sometimes it is possible to face a timeout issue when trying to push huge files. The steps below can be followed:

1. Connect to some EC2 server inside the same Amazon S3 region: us-east-2
2. Copy the file to the server using scp:

```
$ scp -i mykey.pem somefile.txt remote_username@machine.us-east-2.compute.
↪amazonaws.com:/tmp
```

3. Clone the repository as usual (make sure git and git-lfs are properly installed)
4. Just push the files as usual

3.5 Keeping Production Data out of Master Branch

Production versus test data. Any branch beginning with “production” or “application” will contain data in the “production” subfolder. This data should stay in that branch and not be merged into master. To keep the data out, the easiest practice is to simply keep merges one-way from master into the production branch and not vice versa.

However, sometimes troubleshooting / debugging / development happens on a production branch. The cleanest way to get changes to source code or other non-production files back into master is the following.

Checkout your production branch:

```
git checkout production-branch
```

Bring branch even with master:

```
git merge master
```

Resolve conflicts if needed

Capture the files that are different now between production and master:

```
git diff --name-only HEAD master > diff-with-master.txt
```

You have created a file “diff-with-master.txt” containing a listing of every file that is different.

IMPORTANT!!!! – Edit the file diff-with-master.txt and remove all production-related data (this typically will be all files underneath “production” sub-directory).

Checkout master:

```
git checkout master
```

Create a new branch off of master, this is where you will stage the files to then merge back into master:

```
git checkout -b new-branch-with-changes-4ci
```

Do a file by file checkout of all differing files from production branch onto master:

```
cat diff-with-master.txt | xargs git checkout production-branch --
```

Note, if any of our diffs include the deletion of a file on your production branch, then you will need to remove (i.e. with “git remove” these before you do the above “checkout” step and you should also remove them from the diff-with-master.txt”). If you don’t do this, you will see an error message (“did not match any file(s) known to git.”) and the checkout command will not be completed.

Finally, commit the files that were checked out of the production branch, push, and go create your pull request!

3.6 Automated Cloud Deployment

This functionality is available for core BEAM development team with Amazon Web Services access privileges. Please contact [Colin](#) for access to capability.

To run a BEAM simulation or experiment on amazon ec2, use following command with some optional parameters:

```
gradle deploy -P[beamConfigs | beamExperiments]=config-or-experiment-file
```

The command will start an ec2 instance based on the provided configurations and run all simulations in serial. At the end of each simulation/experiment, outputs are uploaded to a public Amazon S3 [bucket](#). To run each simulation/experiment parallel on separate instances, set *beamBatch* to false. For customized runs, you can also use following parameters that can be specified from command line:

- **beamBranch:** To specify the branch for simulation, current source branch will be used as default branch.
- **beamCommit:** The commit SHA to run simulation. use *HEAD* if you want to run with latest commit, default is *HEAD*.
- **beamConfigs:** A comma , separated list of *beam.conf* files. It should be relative path under the project home. You can create branch level defaults by specifying the branch name with *.configs* suffix like *master.configs*. Branch level default will be used if *beamConfigs* is not present.
- **beamExperiments:** A comma , separated list of *experiment.yml* files. It should be relative path under the project home. You can create branch level defaults same as configs by specifying the branch name with *.experiments* suffix like *master.experiments*. Branch level default will be used if *beamExperiments* is not present. *beamConfigs* has priority over this, in other words, if both are provided then *beamConfigs* will be used.
- **beamBatch:** Set to *false* in case you want to run as many instances as number of config/experiment files. Default is *true*.
- **region:** Use this parameter to select the AWS region for the run, all instances would be created in specified region. Default *region* is *us-east-2*.
- **shutdownWait:** As simulation ends, ec2 instance would automatically terminate. In case you want to use the instance, please specify the wait in minutes, default wait is 30 min.

If any of the above parameter is not specified at the command line, then default values are assumed for optional parameters. These default values are specified in [gradle.properties](#) file.

To run a batch simulation, you can specify multiple configuration files separated by commas:

```
gradle deploy -PbeamConfigs=test/input/beamville/beam.conf,test/input/sf-light/sf-  
↪light.conf
```

Similarly for experiment batch, you can specify comma-separated experiment files:

```
gradle deploy -PbeamExperiments=test/input/beamville/calibration/transport-cost/
↪experiments.yml,test/input/sf-light/calibration/transport-cost/experiments.yml
```

For demo and presentation material, please follow the [link](#) on google drive.

3.6.1 AWS EC2 Start

To maintain ec2 instances, there are some utility tasks that reduce operation cost tremendously. You can start already available instances using a simple *start* gradle task under aws module. You can specify one or more instance ids by a comma saturated list as *instanceIds* argument. Below is syntax to use the command:

```
cd aws
gradle start -PinstanceIds=<InstanceID1>[,<InstanceID2>]
```

As a result of task, instance DNS would be printed on the console.

3.6.2 AWS EC2 Stop

Just like starting instance, you can also stop already running instances using a simple *stop* gradle task under aws module. You can specify one or more instance ids by a comma saturated list as *instanceIds* argument. Below is syntax to use the command:

```
cd aws
gradle stop -PinstanceIds=<InstanceID1>[,<InstanceID2>]
```

3.7 Performance Monitoring

Beam uses [Kamon](#) as a performance monitoring framework. It comes with a nice API to instrument your application code for metric recoding. Kamon also provide many different pingable recorders like Log Reporter, StatsD, InfluxDB etc. You can configure your desired recorder with project configurations under Kamon/metrics section. When you start the application it will measure the instrumented components and recorder would publish either to console or specified backend where you can monitor/analyse the metrics.

3.7.1 Beam Metrics Utility (*MetricsSupport*)

Beam provides metric utility as part of performance monitoring framework using Kamon API. It makes developers life very easy, all you need is to extend your component from *beam.sim.metrics.MetricsSupport* trait and call your desired utility. As you extend the trait, it will add some handy entity recorder methods in your component, to measure the application behaviour. By using *MetricSupport* you measure following different metricises.

- Count occurrences or number of invocation:

```
countOccurrence("number-of-routing-requests", Metrics.VerboseLevel)
```

In this example first argument of *countOccurrence* is the name of entity you want to record and second is the metric level. It is the simplest utility and just counts and resets to zero upon each flush. you can use it for counting errors or occurrences of specifics events.

- Execution time of some expression, function call or component:

```
latency("time-to-calculate-route", Metrics.RegularLevel) {
    calcRoute(request)
}
```

In this snippet, first two arguments are same as of *countOccurrence*. Next, it takes the actual piece of code/expression for which you want to measure the execution time/latency. In the example above we are measuring the execution time to calculate a router in *R5RoutingWorker*, we named the entity as “request-router-time” and set metric level to *Metrics.RegularLevel*. When this method executes your entity recorder record the metrics and log with provided name.

3.7.2 Beam Metrics Configuration

After instrumenting your code you need configure your desired metric level, recorder backends and other Kamon configurations in your project configuration file (usually beam.conf). Update your metrics configurations as below:

```
beam.metrics.level = "verbose"

kamon {
  trace {
    level = simple-trace
  }

  metric {
    #tick-interval = 5 seconds
    filters {
      trace.includes = [ "**" ]

      akka-actor {
        includes = [ "beam-actor-system/user/router/**", "beam-actor-system/
↪user/worker-*" ]
        excludes = [ "beam-actor-system/system/**", "beam-actor-system/user/
↪worker-helper" ]
      }

      akka-dispatcher {
        includes = [ "beam-actor-system/akka.actor.default-dispatcher" ]
      }
    }
  }

  statsd {
    hostname = 127.0.0.1 # replace with your container in case local loop didn't
↪work
    port = 8125
  }

  influxdb {
    hostname = 18.216.21.254 # specify InfluxDB server IP
    port = 8089
    protocol = "udp"
  }

  modules {
    #kamon-log-reporter.auto-start = yes
    #kamon-statsd.auto-start = yes
    #kamon-influxdb.auto-start = yes
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

Make sure to update the **host** and **port** for StatsD or InfluxDB (either one(or both) of them you are using) with its relevant the server IP address in the abode config.

Other then IP address you also need to confirm few thing in your environment like.

- beam.metrics.level would not be pointing to the value *off*.
- kamon-statsd.auto-start = yes, under kamon.modules.
- build.gradle(Gradle build script) has kamon-statsd, kamon-influxdb or kamon-log-reporter available as dependencies, based on your kamon.modules settings and desired backend/logger.

3.7.3 Setup Docker as Metric Backend

Kamon's [StatsD](#) reporter enables beam to publish matrices to a verity of backends. [Graphite](#) as the StatsD backend and [Grafana](#) to create beautiful dashboards build a very good monitoring ecosystem. To make environment up and running in a few minutes, use Kamon's provided docker image (beam dashboard need to import) from [docker hub](#) or build using Dockerfile and supporting configuration files available in metrics directory under beam root. All you need is to install few prerequisite like docker, docker-compose, and make. To start a container you just need to run the following command in metrics directory (available at root of beam project):

```
$ make up
```

With the docker container following services start and exposes the listed ports:

- 80: the Grafana web interface.
- 81: the Graphite web port
- 2003: the Graphite data port
- 8125: the StatsD port.
- 8126: the StatsD administrative port.

Now your docker container is up and required components are configured, all you need to start beam simulation. As simulation starts, kamon would load its modules and start publishing metrics to the StatsD server (running inside the docker container).

In your browser open <http://localhost:80> (or with IP you located in previous steps). Login with the default username (admin) and password (admin), open existing beam dashboard (or create a new one).

If you get the docker image from docker hub, you need to import the beam dashboard from metrics/grafana/dashboards directory.

- To import a dashboard open dashboard search and then hit the import button.
- From here you can upload a dashboard json file, as upload complete the import process will let you change the name of the dashboard, pick graphite as data source.
- A new dashboard will appear in dashboard list.

Open beam dashboard (or what ever the name you specified while importing) and start monitoring different beam module level matrices in a nice graphical interface.

To view the container log:

```
$ make tail
```

To stop the container:

```
$ make down
```

Cloud visualization services become more popular nowadays and save much effort and energy to prepare an environment. In future we are planing to use [Datadog](#) (a cloud base monitoring and analytic platform) with beam. [Kamon Datadog integration](#) is the easiest way to have something (nearly) production ready.

How to get Docker IP?

Docker with VirtualBox on macOS/Windows: use docker-machine IP instead of localhost. To find the docker container IP address, first you need to list the containers to get container id using:

```
$ docker ps
```

Then use the container id to find IP address of your container. Run the following command by providing container id in following command by replacing YOUR_CONTAINER_ID:

```
$ docker inspect YOUR_CONTAINER_ID
```

Now at the bottom, under NetworkSettings, locate IP Address of your docker container.

3.8 Tagging Tests for Periodic CI

ScalaTest allows you to define different test categories by tagging your tests. These tags categorise tests in different sets. And later you can filter these set of tests by specifying these tags with your build tasks. Beam also provide a custom tag *Periodic* to mark your tests for periodic CI runs. As you mark the test with this tag, your test would be included automatically into execution set and become the part of next scheduled run. It also be excluded immediately for regular gradle test task and CI. Follow the example below to tag your test with *Periodic* tag:

```
behavior of "Trajectory"
  it should "interpolate coordinates" taggedAs Periodic in {
    ...
  }
```

This code marks the test with *com.beam.tags.Periodic* tag. You can also specify multiple tags as a comma separated parameter list in *taggedAs* method. Following code demonstrate the use of multiple tags:

```
"The agentsim" must {
  ...

  "let everybody walk when their plan says so" taggedAs (Periodic, Slow) in {
    ...
  }

  ...
}
```

You can find details about scheduling a continuous integration build under DevOps section [Configure Periodic Jobs](#).

3.9 Instructions for forking BEAM

These instructions are based on [this page](#)

1. Clone BEAM repo

```
git clone https://github.com/LBNL-UCB-STI/beam
cd beam
```

When asked for user name and password for LFS server (<http://52.15.173.229:8080>) enter anything but do not leave them blank.

2. Fetch Git LFS files

```
git lfs fetch origin
```

Many tutorials on cloning Git LFS repos say one should use

```
git lfs fetch --all origin
```

However, in BEAM this represents over 15 GB data and often fails.

3. Add new origin

```
git remote add new-origin <URL to new repo>
```

4. Create internal master branch, master branch will used to track public repo

```
git branch master-internal
git checkout master-internal
```

5. Update .lfsconfig to have only the new LFS repo

```
[lfs] url = <URL to new LFS repo>
```

Note: for Bitbucket, the <URL to new LFS repo> is <URL to new repo>/info/lfs

6. Commit changes

```
git commit --all
```

7. Push to new repo

```
git push new-origin --all
```

There will be errors saying that many files are missing (LFS upload missing objects). That is OK.

Note: As of this writing, the repo has around 250 MB LFS data. However, the push fails if the LFS server sets a low limit on LFS data. For example, it fails for Bitbucket free which sets a limit of 1 GB LFS data

8. Set master-internal as default branch in the repository's website.

9. Clone the new repo

```
git clone <URL to new repo>
cd <folder of new repo>
```

Note: Cloning might take a few minutes since the repo is quite large.

If everything turned out well, the cloning process should not ask for the credentials for BEAM's LFS server (<http://52.15.173.229:8080>).

10. Add public repo as upstream remote

```
git remote add upstream https://github.com/LBNL-UCB-STI/beam
```

11. Set master branch to track public remote and pull latest changes

```
git fetch upstream
git checkout -b master upstream/master
git pull
```

3.10 Scala tips

3.10.1 Scala Collection

Use mutable buffer instead of immutable var:

```
// Before
var buffer = scala.collection.immutable.Vector.empty[Int]
buffer = buffer :+ 1
buffer = buffer :+ 2

// After
val buffer = scala.collection.mutable.ArrayBuffer.empty[Int]
buffer += 1
buffer += 2
```

Additionally note that, for the best performance, use mutable inside of methods, but return an immutable

```
val mutableList = scala.collection.mutable.MutableList(1,2) mutableList += 3 mutableList.toList //returns
scala.collection.immutable.List
```

//or return mutableList but explicitly set the method return type to //a common, assumed immutable one from scala.collection (more dangerous)

Don't create temporary collections, use view:

```
val seq: Seq[Int] = Seq(1, 2, 3, 4, 5)

// Before
seq.map(x => x + 2).filter(x => x % 2 == 0).sum

// After
seq.view.map(x => x + 2).filter(x => x % 2 == 0).sum
```

Don't emulate collectFirst and collect:

```
// collectFirst
// Get first number >= 4
val seq: Seq[Int] = Seq(1, 2, 10, 20)
val predicate: Int => Boolean = (x: Int) => { x >= 4 }

// Before
seq.filter(predicate).headOption

// After
seq.collectFirst { case num if predicate(num) => num }

// collect
// Get first char of string, if it's longer than 3
val s: Seq[String] = Seq("C#", "C++", "C", "Scala", "Haskel")
val predicate: String => Boolean = (s: String) => { s.size > 3 }

// Before
s.filter(predicate).map { s => s.head }

// After
s.collect { case curr if predicate(curr) => curr.head }
```

Prefer nonEmpty over size > 0:

```
//Before
(1 to x).size > 0

//After
(1 to x).nonEmpty

//nonEmpty shortcircuits as soon as the first element is encountered
```

Prefer not to use _1, _2, ... for Tuple to improve readability:

```
// Get odd elements of sequence s
val predicate: Int => Boolean = (idx: Int) => { idx % 2 == 1 }
val s: Seq[String] = Seq("C#", "C++", "C", "Scala", "Haskel")

// Before
s.zipWithIndex.collect {
  case x if predicate(x._2) => x._1 // what is _1 or _2 ??
}

// After
s.zipWithIndex.collect {
  case (s, idx) if predicate(idx) => s
}

// Use destructuring bindings to extract values from tuple
val tuple = ("Hello", 5)

// Before
```

(continues on next page)

(continued from previous page)

```
val str = tuple._1
val len = tuple._2

// After
val (str, len) = tuple
```

Great article about Scala Collection tips and tricks, must read

3.10.2 Use lazy logging

When you log, prefer to use API which are lazy. If you use `scala logging`, you have [it for free](#). When use `ActorLogging` in Akka, you should not use [string interpolation](#), but use method with replacement arguments:

```
// Before
log.debug(s"Hello: $name")

// After
log.debug("Hello: {}", name)
```

BEAM is composed of Actors. Some of these Actors are BeamAgents. BeamAgents inherit the Akka FSM trait which provides a domain-specific language for programming agent actions as a finite state machine.

How are BeamAgents different from Actors?

In general, we reserve “BeamAgent” (also referred to as “Agent”) for entities in the simulation that exhibit agency. I.e. they don’t just change state but they have some degree of control or autonomy over themselves or other Agents.

A Person or a Manager is a Agent, but a Vehicle is only a tool used by Agents, so it is not a BeamAgent in BEAM.

Also, only BeamAgents can schedule callbacks with the BeamAgentScheduler. So any entity that needs to schedule itself (or be scheduled by other entities) to execute some process at a defined time within the simulation should be designed as a BeamAgent.

Programming a BeamAgent involves constructing a finite state machine and the corresponding logic that responds to Akka messages from different states and then optionally transitions between states.

4.1 Person Agents

4.2 Ride Hail Agents

4.3 Transit Driver Agents

Person Agents in BEAM exhibit several within-day behaviors that govern their use of the transportation system.

5.1 Mode Choice

The most prominent behavior is mode choice. Mode choice can be specified either exogenously as a field in the persons plans, or it can be selected during replanning, or it can remain unset and be selected within the day.

Within day mode choice is selected based on the attributes of the first trip of each tour. Once a mode is selected for the tour, the person attempts to stick with that mode for the duration of the tour.

In all cases (whether mode is specified before the day or chosen within the day) person agents use WALK as a fallback option throughout if constraints otherwise prevent their previously determined mode from being possible for any given trip. E.g. if a person is in the middle of a RIDE_HAIL tour, but the Ride Hail Manager is unable to match a driver to the person, then the person will walk.

In BEAM the following modes are considered:

- Walk
- Bike
- Drive (alone)
- Walk to Transit
- Drive to Transit (Park and Ride)
- Ride Hail
- Ride Hail to/from Transit

There are two mode choice models that are possible within BEAM.

5.1.1 Multinomial Logit Mode Choice

The first is a simple multinomial logit choice model that has the following form for modal alternative j :

$$V_j = \text{ASC}_j + \text{Beta}_{\text{cost}} * \text{cost} + \text{Beta}_{\text{time}} * \text{time} + \text{Beta}_{\text{xfer}} * \text{num_transfers}$$

The ASC (alternative specific constant) parameters as well as the Beta parameters can be configured in the BEAM configuration file and default to the following values:

```
beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.cost = -1.0 beam.agentsim.agents.modalBehaviors.mulitnomialLogit.p
= -0.0047 beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.transfer = -
1.4 beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.car_intercept = 0.0
beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.walk_transit_intercept = 0.0
beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.drive_transit_intercept = 0.0
beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.ride_hail_transit_intercept =
0.0 beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.ride_hail_intercept =
0.0 beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.walk_intercept = 0.0
beam.agentsim.agents.modalBehaviors.mulitnomialLogit.params.bike_intercept = 0.0
```

5.1.2 Latent Class Mode Choice

5.2 Parking

In BEAM, parking is issued at the granularity of a Traffic Analysis Zone (TAZ). Upon initialization, parking alternatives are read in from the CSV file listed in the BEAM config parameter *beam.agentsim.taz.parking*. Each row identifies the attributes of a parking alternative for a given TAZ, of which a given combination of attributes should be unique. Parking attributes include the following:

| attribute | values |
|---------------------|--|
| <i>parkingType</i> | Workplace, Public, Residential |
| <i>pricingModel</i> | FlatFee, Block |
| <i>chargingType</i> | NoCharger, Level1, Level2, DCFast, UltraFast |
| <i>numStalls</i> | integer |
| <i>feeInCents</i> | integer |
| <i>reservedFor</i> | Any, RideHailManager |

BEAM agents seek parking mid-tour, from within a leg of their trip. A search is run which starts at the trip destination and expands outward, seeking to find the closest TAZ centers with increasing search radii. Agents will pick the closest and cheapest parking alternative with attributes which match their use case. The location can be overridden for ride hail agents using the config parameter *beam.agentsim.agents.rideHail.refuelLocationType*, which may be set to “AtRequestLocation” or “AtTAZCenter”.

The following should be considered when configuring a set of parking alternatives. The default behavior is to provide a nearly unbounded number of parking stalls for each combination of attributes, per TAZ, for the public, and provide no parking alternatives for ride hail agents. This behavior can be overridden manually by providing replacement values in the parking configuration file. Parking which is *reservedFor* a RideHailManager should only appear as *Workplace* parking. Free parking can be instantiated by setting *feeInCents* to zero. *numStalls* should be non-negative. Charging behavior is currently implemented for ride hail agents only.

the *chargingType* attribute will result in the following charger power in kW:

| <i>chargingType</i> | kW |
|---------------------|-------|
| NoCharger | 0.0 |
| Level1 | 1.5 |
| Level2 | 6.7 |
| DCFast | 50.0 |
| UltraFast | 250.0 |

5.3 Refueling

Event Specifications

For an overview of events, including compatibility with MATSim, see [MATSim Events](#).

The following lists each field in each event with some brief descriptions and contextual information where necessary.

6.1 MATSim Events

The following MATSim events are thrown within the AgentSim:

6.1.1 ActivityStartEvent

- Time - Time of the start of the activity.
- Activity Type - String denoting the type of activity (e.g. “Home” or “Work”)
- Person - Person ID of the person agent engaged in the activity.
- Link - Link ID of the nearest link to the activity location
- Facility - Facility ID (unused in BEAM)

6.1.2 ActivityEndEvent

- Time - Time of the end of the activity.
- Activity Type - String denoting the type of activity (e.g. “Home” or “Work”)
- Person - Person ID of the person agent engaged in the activity.
- Link - Link ID of the nearest link to the activity location
- Facility - Facility ID (unused in BEAM)

6.1.3 PersonDepartureEvent

- Time - Time of the person departure.
- Person - Person ID of the person departing.
- Leg Mode - String denoting the trip mode of the trip to be attempted (trip mode is the overall mode of the trip, which is different than the mode of individual sub-legs of the trip, e.g. a trip with leg mode TRANSIT might have sub-legs of mode WALK, BUS, SUBWAY, WALK).
- Link - Link ID of the nearest link to the departure location.

6.1.4 PersonArrivalEvent

- Time - Time of the person arrival.
- Person - Person ID of the person arriving.
- Leg Mode - String denoting the trip mode of the trip completed (trip mode is the overall mode of the trip, which is different than the mode of individual sub-legs of the trip, e.g. a trip with leg mode TRANSIT might have sub-legs of mode WALK, BUS, SUBWAY, WALK).
- Link - Link ID of the nearest link to the arrival location.

6.1.5 PersonEntersVehicleEvent

- Time - Time of the vehicle entry.
- Person - Person ID of the person entering the vehicle.
- Vehicle - Vehicle ID of the vehicle being entered.

6.1.6 PersonLeavesVehicleEvent

- Time - Time of the vehicle exit.
- Person - Person ID of the person exiting the vehicle.
- Vehicle - Vehicle ID of the vehicle being exited.

6.2 BEAM Events

These events are specific to BEAM and are thrown within the AgentSim:

6.2.1 ModeChoiceEvent

Note that this event corresponds to the moment of choosing a mode, if mode choice is occurring dynamically within the day. If mode choice occurs outside of the simulation day, then this event is not thrown. Also, the time of choosing mode is not always the same as the departure time.

- Time - Time of the mode choice.
- Person - Person ID of the person making the mode choice.
- Mode - The chosen trip mode (e.g. WALK_TRANSIT or CAR)

- Expected Maximum Utility - The logsum from the utility function used to evaluate modal alternatives. If the mode choice model is not a random utility based model, then this will be left blank.
- Location - Link ID of the nearest location.
- Available Alternatives - Comma-separated list of the alternatives considered by the agent during the mode choice process.
- Persona Vehicle Available - Boolean denoting whether this agent had a personal vehicle available to them during the mode choice process.
- Length - the length of the chosen trip in meters.
- Tour index - the index of the chosen trip within the current tour of the agent (e.g. 0 means the first trip of the tour, 1 is the second trip, etc.)

6.2.2 PathTraversalEvent

A Path Traversal is any time a vehicle moves within the BEAM AgentSim.

- Length - Length of the movement in meters.
- Fuel - fuel consumed during the movement in Joules.
- Num Passengers - the number of passengers on board during the vehicle movement (the driver does not count as a passenger).
- Links - Comma-separated list of link IDs indicating the path taken.
- Mode - the sub-leg mode of the traversal (e.g. BUS or CAR or SUBWAY).
- Departure Time - the time of departure.
- Arrival Time - the time of arrival.
- Vehicle - the ID of the vehicle making the movement.
- Vehicle Type - String indicating the type of vehicle.
- Start X - X coordinate of the starting location of the movement. Coordinates are output in WGS (lat/lon).
- Start Y - Y coordinate of the starting location of the movement. Coordinates are output in WGS (lat/lon).
- End X - X coordinate of the ending location of the movement. Coordinates are output in WGS (lat/lon).
- End Y - Y coordinate of the ending location of the movement. Coordinates are output in WGS (lat/lon).
- End Leg Fuel Level - Amount of fuel (in Joules) remaining in the vehicle at the end of the movement.

7.1 Configuration file

The BEAM configuration file controls where BEAM will source input data and the value of parameters. To see an example of the latest version of this file:

<https://github.com/LBNL-UCB-STI/beam/blob/master/test/input/beamville/beam.conf>

As of Fall 2018, BEAM is still under rapid development. So the configuration file will continue to evolve. Particularly, it should be expected that new parameters will be created to control new model features and old configuration options may be modified, simplified, or eliminated.

Furthermore, the BEAM configuration file contains a hybrid between parameters from MATSim (see namespace *matsim* in the config file). Not all of the matsim parameters are used by BEAM. Only the specific MATSim parameters described in this document are relevant. Modifying the other parameters will have no impact. In future releases of BEAM, the irrelevant parameters will be removed.

In order to see example configuration options for a particular release of BEAM replace *master* in the above URL with the version number, e.g. for Version v0.6.2 go to this link:

<https://github.com/LBNL-UCB-STI/beam/blob/v0.6.2/test/input/beamville/beam.conf>

BEAM follows the [MATSim convention](#) for most of the inputs required to run a simulation, though specifying the road network and transit system is based on the [R5 requirements](#). Refer to these external documentation for details on the following inputs.

- The person population and corresponding attributes files (e.g. *population.xml* and *populationAttributes.xml*)
- The household population and corresponding attributes files (e.g. *households.xml* and *householdAttributes.xml*)
- A directory containing network and transit data used by R5 (e.g. *r5/*)
- The open street map network (e.g. *r5/beamville.osm*)
- GTFS archives, one for each transit agency (e.g. *r5/bus.zip*)

7.1.1 Config Options

The following is a list of the most commonly used configuration options in approximate order of appearance in the beamville example config file (order need not be preserved so it is ok to rearrange options). A complete listing will be added to this documentation soon

General parameters:

```
beam.agentsim.simulationName = "beamville"
beam.agentsim.numAgents = 100
beam.agentsim.thresholdForWalkingInMeters = 1000
beam.agentsim.thresholdForMakingParkingChoiceInMeters = 100
beam.agentsim.schedulerParallelismWindow = 30
beam.agentsim.timeBinSize = 3600
```

- **simulationName:** Used as a prefix when creating an output directory to store simulation results.
- **numAgents:** This will limit the number of PersonAgents created in the simulation agents will be . Note that the number of agents is also limited by the total number of “person” elements in the population file specified by *matsim.modules.plans.inputPlansFile*. In other words, if there are 100 people in the plans and numAgents is set to 50, then 50 PersonAgents will be created. If numAgents is ≥ 100 , then 100 PersonAgents will be created. Sampling to a smaller number of agents is accomplished by sampling full households until the desired number of PersonAgents is reached. This keeps the household structure intact.
- **thresholdForWalkingInMeters:** Used to determine whether a PersonAgent needs to route a walking path through the network to get to a parked vehicle. If the vehicle is closer than thresholdForWalkingInMeters in Euclidean distance, then the walking trip is assumed to be instantaneous. Note, for performance reasons, we do not recommend values for this threshold less than 100m.
- **thresholdForMakingParkingChoiceInMeters:** Similar to thresholdForWalkingInMeters, this threshold determines the point in a driving leg when the PersonAgent initiates the parking choice processes. So for 1000m, the agent will drive until she is ≤ 1 km from the destination and then seek a parking space.
- **schedulerParallelismWindow:** This controls the discrete event scheduling window used by BEAM to achieve within-day parallelism. The units of this parameter are in seconds and the larger the window, the better the performance of the simulation, but the less chronologically accurate the results will be.
- **timeBinSize:** For most auto-generated output graphs and tables, this parameter will control the resolution of time-varying outputs.

Mode choice parameters:

```
beam.agentsim.agents.modalBehaviors.modeChoiceClass = "ModeChoiceMultinomialLogit"
beam.agentsim.agents.modalBehaviors.defaultValueOfTime = 8.0
beam.agentsim.agents.modalBehaviors.multinomialLogit.params.transfer = -1.4
beam.agentsim.agents.modalBehaviors.multinomialLogit.params.car_intercept = 0.0
beam.agentsim.agents.modalBehaviors.multinomialLogit.params.walk_transit_intercept =
↪ 0.0
beam.agentsim.agents.modalBehaviors.multinomialLogit.params.drive_transit_intercept =
↪ 2.0
beam.agentsim.agents.modalBehaviors.multinomialLogit.params.ride_hail_transit_
↪ intercept = 0.0
beam.agentsim.agents.modalBehaviors.multinomialLogit.params.ride_hail_intercept = -1.0
beam.agentsim.agents.modalBehaviors.multinomialLogit.params.walk_intercept = -3.0
beam.agentsim.agents.modalBehaviors.multinomialLogit.params.bike_intercept = 0.0
beam.agentsim.agents.modalBehaviors.lccm.paramFile = ${beam.inputDirectory}"/lccm-
↪ long.csv"
#Toll params
beam.agentsim.toll.file=${beam.inputDirectory}"/toll-prices.csv"
```


- `modeChoiceClass`: Selects the choice algorithm to be used by agents to select mode when faced with a choice. Default of `ModeChoiceMultinomialLogit` is recommended but other algorithms include `ModeChoiceMultinomialLogit` `ModeChoiceTransitIfAvailable` `ModeChoiceDriveIfAvailable` `ModeChoiceRideHailIfAvailable` `ModeChoiceUniformRandom` `ModeChoiceLCCM`.
- `defaultValueOfTime`: This value of time is used by the `ModeChoiceMultinomialLogit` choice algorithm unless the value of time is specified in the `populationAttributes` file.
- `params.transfer`: Constant utility (where 1 util = 1 dollar) of making transfers during a transit trip.
- `params.car_intercept`: Constant utility (where 1 util = 1 dollar) of driving.
- `params.walk_transit_intercept`: Constant utility (where 1 util = 1 dollar) of walking to transit.
- `params.drive_transit_intercept`: Constant utility (where 1 util = 1 dollar) of driving to transit.
- `params.ride_hail_transit_intercept`: Constant utility (where 1 util = 1 dollar) of taking ride hail to/from transit.
- `params.ride_hail_intercept`: Constant utility (where 1 util = 1 dollar) of taking ride hail.
- `params.walk_intercept`: Constant utility (where 1 util = 1 dollar) of walking.
- `params.bike_intercept`: Constant utility (where 1 util = 1 dollar) of biking.
- `lccm.paramFile`: if `modeChoiceClass` is set to `ModeChoiceLCCM` this must point to a valid file with LCCM parameters. Otherwise, this parameter is ignored.
- `toll.file`: File path to a file with static road tolls. Note, this input will change in future BEAM release where time-varying tolls will be possible.

Vehicles and Population:

```
#BeamVehicles Params
beam.agentsim.agents.vehicles.beamFuelTypesFile = ${beam.inputDirectory}"/
↳beamFuelTypes.csv"
beam.agentsim.agents.vehicles.beamVehicleTypesFile = ${beam.inputDirectory}"/
↳vehicleTypes.csv"
beam.agentsim.agents.vehicles.beamVehiclesFile = ${beam.inputDirectory}"/vehicles.csv"
```

- `useBikes`: simple way to disable biking, set to true if vehicles file does not contain any data on biking.
- `beamFuelTypesFile`: configure fuel fuel pricing.
- `beamVehicleTypesFile`: configure vehicle properties including seating capacity, length, fuel type, fuel economy, and refueling parameters.
- `beamVehiclesFile`: replacement to legacy MATSim `vehicles.xml` file. This must contain an Id and vehicle type for every vehicle id contained in `households.xml`.

TAZs, Scaling, and PhysSim Tuning:

```
#TAZ params
beam.agentsim.taz.file=${beam.inputDirectory}"/taz-centers.csv"
beam.agentsim.taz.parking = ${beam.inputDirectory}"/parking/taz-parking-default.csv"
# Scaling and Tuning Params
beam.agentsim.tuning.transitCapacity = 0.1
beam.agentsim.tuning.transitPrice = 1.0
beam.agentsim.tuning.tollPrice = 1.0
beam.agentsim.tuning.rideHailPrice = 1.0
# PhysSim Scaling Params
beam.physsim.flowCapacityFactor = 0.0001
beam.physsim.storageCapacityFactor = 0.0001
beam.physsim.writeMATSimNetwork = false
```

(continues on next page)

(continued from previous page)

```
beam.physsim.ptSampleSize = 1.0
beam.physsim.jdeqsim.agentSimPhysSimInterfaceDebugger.enabled = false
beam.physsim.skipPhysSim = false
```

- `agentsim.taz.file`: path to a file specifying the centroid of each TAZ. For performance BEAM approximates TAZ boundaries based on a nearest-centroid approach. The area of each centroid (in m^2) is also necessary to approximate average travel distances within each TAZ (used in parking choice process).
- `taz.parking`: path to a file specifying the parking and charging infrastructure. If any TAZ contained in the taz file is not specified in the parking file, then unlimited free parking is assumed.
- `tuning.transitCapacity`: Scale the number of seats per transit vehicle. . . actual seats are rounded to nearest whole number. Applies uniformly to all transit vehicles.
- `tuning.transitPrice`: Scale the price of riding on transit. Applies uniformly to all transit trips.
- `tuning.tollPrice`: Scale the price to cross tolls.
- `tuning.rideHailPrice`: Scale the price of ride hailing. Applies uniformly to all trips and is independent of `defaultCostPerMile` and `defaultCostPerMinute` described above. I.e. $price = (costPerMile + costPerMinute) * rideHailPrice$
- `physsim.flowCapacityFactor`: Flow capacity parameter used by JDEQSim for traffic flow simulation.
- `physsim.storageCapacityFactor`: Storage capacity parameter used by JDEQSim for traffic flow simulation.
- `physsim.writeMATSimNetwork`: A copy of the network used by JDEQSim will be written to outputs folder (typically only needed for debugging).
- `physsim.ptSampleSize`: A scaling factor used to reduce the seating capacity of all transit vehicles. This is typically used in the context of running a partial sample of the population, it is advisable to reduce the capacity of the transit vehicles, but not necessarily proportionately. This should be calibrated.
- `agentSimPhysSimInterfaceDebugger.enabled`: Enables special debugging output.
- `skipPhysSim`: Turns off the JDEQSim traffic flow simulation. If set to true, then network congestion will not change from one iteration to the next. Typically this is only used for debugging issues that are unrelated to the `physsim`.

Warm Mode:

```
#####
# Warm Mode
#####
beam.warmStart.enabled = false
#PATH TYPE OPTIONS: PARENT_RUN, ABSOLUTE_PATH
#PARENT_RUN: can be a director or zip archive of the output directory (e.g. like what_
↳get's stored on S3). We should also be able to specify a URL to an S3 output.
#ABSOLUTE_PATH: a directory that contains required warm stats files (e.g. linkstats_
↳and eventually a plans).
beam.warmStart.pathType = "PARENT_RUN"
beam.warmStart.path = "https://s3.us-east-2.amazonaws.com/beam-outputs/run149-base__
↳2018-06-27_20-28-26_2a2e2bd3.zip"
```

- `warmStart.enabled`: Allows you to point to the output of a previous BEAM run and the network travel times and final plan set from that run will be loaded and used to start a new BEAM run.
- `beam.warmStart.pathType`: See above for descriptions.
- `beam.warmStart.path`: path to the outputs to load. Can be a path on the local computer or a URL in which case outputs will be downloaded.

Ride hail management:

```
#####
# RideHail
#####
# Ride Hailing General Params
beam.agentsim.agents.rideHail.numDriversAsFractionOfPopulation=0.1
beam.agentsim.agents.rideHail.defaultCostPerMile=1.25
beam.agentsim.agents.rideHail.defaultCostPerMinute=0.75
beam.agentsim.agents.rideHail.vehicleTypeId="BEV"
beam.agentsim.agents.rideHail.refuelThresholdInMeters=5000.0
beam.agentsim.agents.rideHail.refuelLocationType="AtRequestLocation"
# SurgePricing parameters
beam.agentsim.agents.rideHail.surgePricing.surgeLevelAdaptionStep=0.1
beam.agentsim.agents.rideHail.surgePricing.minimumSurgeLevel=0.1

# priceAdjustmentStrategy(KEEP_PRICE_LEVEL_FIXED_AT_ONE | CONTINUES_DEMAND_SUPPLY_
↪MATCHING)
beam.agentsim.agents.rideHail.surgePricing.priceAdjustmentStrategy="KEEP_PRICE_LEVEL_
↪FIXED_AT_ONE"

beam.agentsim.agents.rideHail.rideHailManager.radiusInMeters=5000

# initialLocation(HOME | UNIFORM_RANDOM | ALL_AT_CENTER | ALL_IN_CORNER)
beam.agentsim.agents.rideHail.initialLocation.name="HOME"
beam.agentsim.agents.rideHail.initialLocation.home.radiusInMeters=10000

# allocationManager(DEFAULT_MANAGER | REPOSITIONING_LOW_WAITING_TIMES | EV_MANAGER)
beam.agentsim.agents.rideHail.allocationManager.name="EV_MANAGER"
beam.agentsim.agents.rideHail.allocationManager.timeoutInSeconds=300
beam.agentsim.agents.rideHail.allocationManager.randomRepositioning.
↪repositioningShare=0.2

beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪repositionCircleRadiusInMeters=3000.0
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪minimumNumberOfIdlingVehiclesThreshholdForRepositioning=1
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪percentageOfVehiclesToReposition=1.0
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪timeWindowSizeInSecForDecidingAboutRepositioning=1200
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪allowIncreasingRadiusIfDemandInRadiusLow=true
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪minDemandPercentageInRadius=0.1
# repositioningMethod(TOP_SCORES | KMEANS)
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪repositioningMethod="TOP_SCORES"
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪keepMaxTopNScores=5
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪minScoreThresholdForRepositioning=0.00001
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪distanceWeight=0.01
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪waitingTimeWeight=4.0
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.
↪demandWeight=4.0
```

(continues on next page)

(continued from previous page)

```
beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes.  
↪produceDebugImages=true  
  
beam.agentsim.agents.rideHail.iterationStats.timeBinSizeInSec=3600
```

- `numDriversAsFractionOfPopulation` - Defines the # of ride hailing drivers to create, this ration is multiplied by the parameter `beam.agentsim.numAgents` to determine the actual number of drivers to create. Drivers begin the simulation located at or near the homes of existing agents, uniformly distributed.
- `defaultCostPerMile` - One component of the 2 part price of ride hail calculation.
- `defaultCostPerMinute` - One component of the 2 part price of ride hail calculation.
- `vehicleTypeId`: What vehicle type is used for ride hail vehicles. This is primarily relevant for when allocation-Manager is *EV_MANAGER*.
- `refuelThresholdInMeters`: One the fuel level (state of charge for EVs) of the vehicle falls below the level corresponding to this parameter, the *EV_MANAGER* will dispatch the vehicle to refuel. Note, do not make this value greate than 80% of the total vehicle range to avoid complications associated with EV fast charging.
- `refuelLocationType`: One of *AtRequestLocation* or *AtTAZCenter* which controls whether the vehicle is assumed to charge at the it's present location (*AtRequestLocation*) or whether it will drive to a nearby charging depot (*AtTAZCenter*).
- `allocationManager.name`: Controls whether fleet management is simple (*DEFAULT_MANAGER* for no repositioning, no refueling), includes repositioing (*REPOSITIONING_LOW_WAITING_TIMES*) or includes both repositioning and refueling (*EV_MANAGER*)
- `allocationManager.timeoutInSeconds`: How frequently does the manager make fleet repositioning decisions.
- `beam.agentsim.agents.rideHail.allocationManager.repositionLowWaitingTimes`: All of these parameters control the details of repositioning, more documentation will be posted for these soon.

8.1 File: /modeChoice.csv

Classname: ModeChosenAnalysisObject

| field | description |
|---------------|--|
| iterations | iteration number |
| car | Car chosen as travel mode |
| drive_transit | Drive to transit chosen as travel mode |
| ride_hail | Ride Hail chosen as travel mode |
| walk | Walk chosen as travel mode |
| walk_transit | Walk to transit chosen as travel mode |

8.2 File: /referenceModeChoice.csv

Classname: ModeChosenAnalysisObject

| field | description |
|-------------------|--|
| iterations | Bike chosen as travel mode |
| bike | iteration number |
| car | Car chosen as travel mode |
| drive_transit | Drive to transit chosen as travel mode |
| ride_hail | Ride Hail chosen as travel mode |
| ride_hail_transit | Ride Hail to transit chosen as travel mode |
| walk | Walk chosen as travel mode |
| walk_transit | Walk to transit chosen as travel mode |

8.3 File: /realizedMode.csv

Classname: RealizedModeAnalysisObject

| field | description |
|---------------|--|
| car | Car chosen as travel mode |
| drive_transit | Drive to transit chosen as travel mode |
| other | Other modes of travel chosen |
| ride_hail | Ride Hail chosen as travel mode |
| walk | Walk chosen as travel mode |
| walk_transit | Walk to transit chosen as travel mode |

8.4 File: /rideHailRevenue.csv

Classname: RideHailRevenueAnalysisObject

| field | description |
|-------------|----------------------------------|
| iteration # | iteration number |
| revenue | Revenue generated from ride hail |

8.5 File: /ITERS/it.0/0.averageTravelTimes.csv

Classname: PersonTravelTimeAnalysisObject

| field | description |
|--------|--|
| Mode | Travel mode chosen |
| Hour,* | Average time taken to travel by the chosen mode during the given hour of the day |

8.6 File: /ITERS/it.0/0.energyUse.png.csv

Classname: FuelUsageAnalysisObject

| field | description |
|-------|--|
| Modes | Mode of travel chosen by the passenger |
| Bin_* | Energy consumed by the vehicle while travelling by the chosen mode within the given time bin |

8.7 File: /ITERS/it.0/0.physsimLinkAverageSpeedPercentage.csv

Classname: PhyssimCalcLinkSpeedStatsObject

| field | description |
|------------------|--|
| Bin | A given time slot within a day |
| AverageLinkSpeed | The average speed at which a vehicle can travel across the network during the given time bin |

8.8 File: /ITERS/it.0/0.physsimFreeFlowSpeedDistribution.csv

Classname: PhyssimCalcLinkSpeedDistributionStatsObject

| field | description |
|----------------------------|---|
| freeSpeedInMetersPerSecond | The possible full speed at which a vehicle can drive through the given link (in m/s) |
| numberOfLinks | Total number of links in the network that allow vehicles to travel with speeds up to the given free speed |
| linkEfficiencyInPercentage | Average speed efficiency recorded by the the given network link in a day |
| numberOfLinks | Total number of links having the corresponding link efficiency |

8.9 File: /ITERS/it.0/0.rideHailWaitingStats.csv

Classname: RideHailWaitingAnalysisObject

| field | description |
|--------------|--|
| Waiting Time | The time spent by a passenger waiting for a ride hail |
| Hour | Hour of the day |
| Count | Frequencies of times spent waiting for a ride hail during the entire day |

8.10 File: /ITERS/it.0/0.rideHailIndividualWaitingTimes.csv

Classname: RideHailWaitingAnalysisObject

| field | description |
|----------------------|---|
| timeOfDayInSeconds | Time of a day in seconds |
| personId | Unique id of the passenger travelling by the ride hail |
| rideHailVehicleId | Unique id of the ride hail vehicle |
| waitingTimeInSeconds | Time spent by the given passenger waiting for the arrival of the given ride hailing vehicle |

8.11 File: /ITERS/it.0/0.rideHailSurgePriceLevel.csv

Classname: GraphSurgePricingObject

| field | description |
|------------|--|
| PriceLevel | Travel fare charged by the ride hail in the given hour |
| Hour | Hour of the day |

8.12 File: /ITERS/it.0/0.rideHailRevenue.csv

Classname: GraphSurgePricingObject

| field | description |
|---------|---|
| Revenue | Revenue earned by ride hail in the given hour |
| Hour | Hour of the day |

8.13 File: /ITERS/it.0/0.tazRideHailSurgePriceLevel.csv

Classname: GraphSurgePricingObject

| field | description |
|----------|---|
| TazId | TAZ id |
| DataType | Type of data , can be “priceLevel” or “revenue” |
| Value | Value of the given data type , can indicate either price Level or revenue earned by the ride hail in the given hour |
| Hour | Hour of the day |

8.14 File: /ITERS/it.0/0.rideHailWaitingSingleStats.csv

Classname: RideHailingWaitingSingleAnalysisObject

| field | description |
|------------------|--|
| WaitingTime(sec) | Time spent by a passenger on waiting for a ride hail |
| Hour* | Hour of the day |

8.15 File: /ITERS/it.0/0.rideHailInitialLocation.csv

Classname: BeamMobsim

| field | description |
|-----------------|---|
| rideHailAgentID | Unique id of the given ride hail agent |
| xCoord | X co-ordinate of the starting location of the ride hail |
| yCoord | Y co-ordinate of the starting location of the ride hail |

8.16 File: /stopwatch.txt

Classname: StopWatchOutput

| field | description |
|--------------------------------|---|
| Iteration | Iteration number |
| BEGIN iteration | Begin time of the iteration |
| BEGIN iterationStartsListeners | Time at which the iteration start event listeners started |
| END iterationStartsListeners | Time at which the iteration start event listeners ended |
| BEGIN replanning | Time at which the replanning event started |
| END replanning | Time at which the replanning event ended |
| BEGIN beforeMobsimListeners | Time at which the beforeMobsim event listeners started |
| BEGIN dump all plans | Begin dump all plans |
| END dump all plans | End dump all plans |
| END beforeMobsimListeners | Time at which the beforeMobsim event listeners ended |
| BEGIN mobsim | Time at which the mobsim run started |
| END mobsim | Time at which the mobsim run ended |
| BEGIN afterMobsimListeners | Time at which the afterMobsim event listeners started |
| END afterMobsimListeners | Time at which the afterMobsim event listeners ended |
| BEGIN scoring | Time at which the scoring event started |
| END scoring | Time at which the scoring event ended |
| BEGIN iterationEndsListeners | Time at which the iteration ends event listeners ended |
| BEGIN compare with counts | Time at which compare with counts started |
| END compare with counts | Time at which compare with counts ended |
| END iteration | Time at which the iteration ended |

8.17 File: /scorestats.txt

Classname: ScoreStatsOutput

| field | description |
|---------------|---|
| ITERATION | Iteration number |
| avg. EXECUTED | Average of the total execution time for the given iteration |
| avg. WORST | Average of worst case time complexities for the given iteration |
| avg. AVG | Average of average case time complexities for the given iteration |
| avg. BEST | Average of best case time complexities for the given iteration |

8.18 File: /summaryStats.txt

Classname: SummaryStatsOutput

8.19 File: /ITERS/it.0/0.countsCompare.txt

Classname: CountsCompareOutput

| field | description |
|---------------------------|--|
| Link Id | Iteration number |
| Count | Time taken by the agent to travel in a crowded transit |
| Station Id | Amount of diesel consumed in megajoule |
| Hour | Amount of food consumed in megajoule |
| MATSIM volumes | Amount of electricity consumed in megajoule |
| Relative Error | Amount of gasoline consumed in megajoule |
| Normalized Relative Error | Time at which the beforeMobsim event listeners started |
| GEH | GEH |

8.20 File: /ITERS/it.0/0.events.csv

Classname: EventOutput

| field | description |
|--------------------------|---|
| person | Person(Agent) Id |
| vehicle | vehicle id |
| time | Start time of the vehicle |
| type | Type of the event |
| fuel | Type of fuel used in the vehicle |
| duration | Duration of the travel |
| cost | Cost of travel |
| location.x | X co-ordinate of the location |
| location.y | Y co-ordinate of the location |
| parking_type | Parking type chosen by the vehicle |
| pricing_model | Pricing model |
| charging_type | Charging type of the vehicle |
| parking_taz | Parking TAZ |
| distance | Distance between source and destination |
| location | Location of the vehicle |
| mode | Mode of travel |
| currentTourMode | Current tour mode |
| expectedMaximumUtility | Expected maximum utility of the vehicle |
| availableAlternatives | Available alternatives for travel for the passenger |
| personalVehicleAvailable | Whether the passenger possesses a personal vehicle |
| tourIndex | Tour index |
| facility | Facility availed by the passenger |
| departTime | Time of departure of the vehicle |
| originX | X ordinate of the passenger origin point |
| originY | Y ordinate of the passenger origin point |
| destinationX | X ordinate of the passenger destination point |
| destinationY | Y ordinate of the passenger destination point |
| fuelType | Fuel type of the vehicle |
| num_passengers | Num of passengers travelling in the vehicle |
| links | Number of links in the network |
| departure_time | Departure time of the vehicle |
| arrival_time | Arrival time of the vehicle |
| vehicle_type | Type of vehicle |
| capacity | Total capacity of the vehicle |

Continued on next page

Table 1 – continued from previous page

| field | description |
|--------------------|--|
| start.x | X ordinate of the start point |
| start.y | Y ordinate of the start point |
| end.x | X ordinate of the vehicle end point |
| end.y | Y ordinate of the vehicle end point |
| end_leg_fuel_level | Fuel level at the end of the travel |
| seating_capacity | Seating capacity of the vehicle |
| costType | Type of cost of travel incurred on the passenger |

8.21 File: /ITERS/it.0/0.legHistogram.txt

Classname: LegHistogramOutput

| field | description |
|--------------------------|---|
| time | Time |
| time | Time |
| departures_all | Total number of departures on all modes |
| arrivals_all | Total number of arrivals on all modes |
| duration | Duration of travel |
| stuck_all | Total number of travels that got stuck on all modes |
| en-route_all | Total number of travels by all modes |
| departures_car | Total number of departures by car |
| arrivals_car | Total number of departures by car |
| stuck_car | Total number of travels that got stuck while travelling by car |
| en-route_car | Total number of travels made by car |
| departures_drive_transit | Total number of departures by drive to transit |
| arrivals_drive_transit | Total number of arrivals by drive to transit |
| stuck_drive_transit | Total number of travels that got stuck while travelling by drive to transit |
| en-route_drive_transit | Total number of travels made by drive to transit |
| departures_ride_hail | Total number of departures by ride hail |
| arrivals_ride_hail | Total number of arrivals by ride hail |
| stuck_ride_hail | Total number of travels that got stuck while travelling by ride hail |
| en-route_ride_hail | Total number of travels made by ride hail |
| departures_walk | Total number of departures on foot |
| arrivals_walk | Total number of arrivals on foot |
| stuck_walk | Total number of travels that got stuck while travelling on foot |
| en-route_walk | Total number of travels made on foot |
| departures_walk_transit | Total number of departures by walk to transit |
| arrivals_walk_transit | Total number of arrivals by walk to transit |
| stuck_walk_transit | Total number of travels that got stuck while travelling by walk to transit |
| en-route_walk_transit | Total number of travels made by walk to transit |

8.22 File: /ITERS/it.0/0.rideHailTripDistance.csv

Classname: RideHailTripDistanceOutput

| field | description |
|---------------|---|
| hour | Hour of the day |
| numPassengers | Number of passengers travelling in the ride hail |
| vkt | Total number of kilometers travelled by the ride hail vehicle |

8.23 File: /ITERS/it.0/0.tripDuration.txt

Classname: TripDurationOutput

| field | description |
|---------|-------------|
| pattern | Pattern |
| (5*i)+ | Value |

8.24 File: /ITERS/it.0/0.biasErrorGraphData.txt

Classname: BiasErrorGraphDataOutput

| field | description |
|---------------------|---------------------|
| hour | Hour of the day |
| mean relative error | Mean relative error |
| mean bias | Mean bias value |

8.25 File: /ITERS/it.0/0.biasNormalizedErrorGraphData.txt

Classname: BiasNormalizedErrorGraphDataOutput

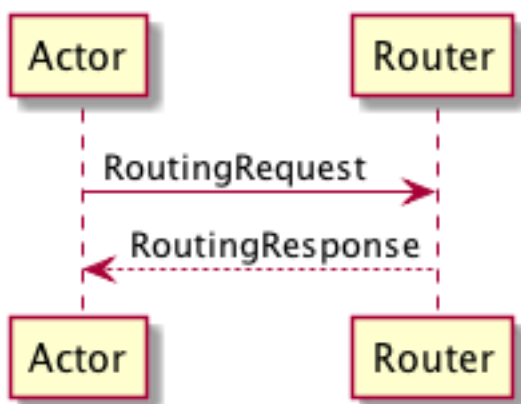
| field | description |
|--------------------------------|--------------------------------|
| hour | Hour of the day |
| mean normalized relative error | Mean normalized relative error |
| mean bias | Mean bias value |

Because BEAM is implemented using the Actor framework and simulations are executed asynchronously, there are many communications protocols between Actors and Agents that must be specified and followed. The following describes the key protocols with diagrams and narrative.

9.1 Trip Planning

9.1.1 RoutingRequests

One of the more familiar protocols, any Actor can consult the router service for routing information by sending a `RoutingRequest` and receiving a `RoutingResponse`.



The `RoutingRequest` message contains:

- Departure Window
- Origin / Destination
- Transit Modes to Consider

- Vehicles to Consider (their Id, Location, Mode)
- The Id of the Person for whom the request is ultimately made

The RoutingResponse message contains:

- A vector of EmbodiedBeamTrips

EmbodiedBeamTrips contain:

- A trip classifier (i.e. the overall mode for the trip which is restricted to WALK, BIKE, CAR, RIDE_HAIL, TRANSIT)
- A vector of EmbodiedBeamLegs

EmbodiedBeamLegs contain:

- A BeamLeg
- A BeamVehicle Id
- As Driver Boolean
- Optional PassengerSchedule
- Cost
- UnbecomeDriveOnCompletion Boolean

BeamLegs contain:

- Start time
- Mode (this is a more specific mode for Transit, e.g. SUBWAY or BUS)
- Duration
- BeamPath containing the path used through the network.

BeamPaths contain:

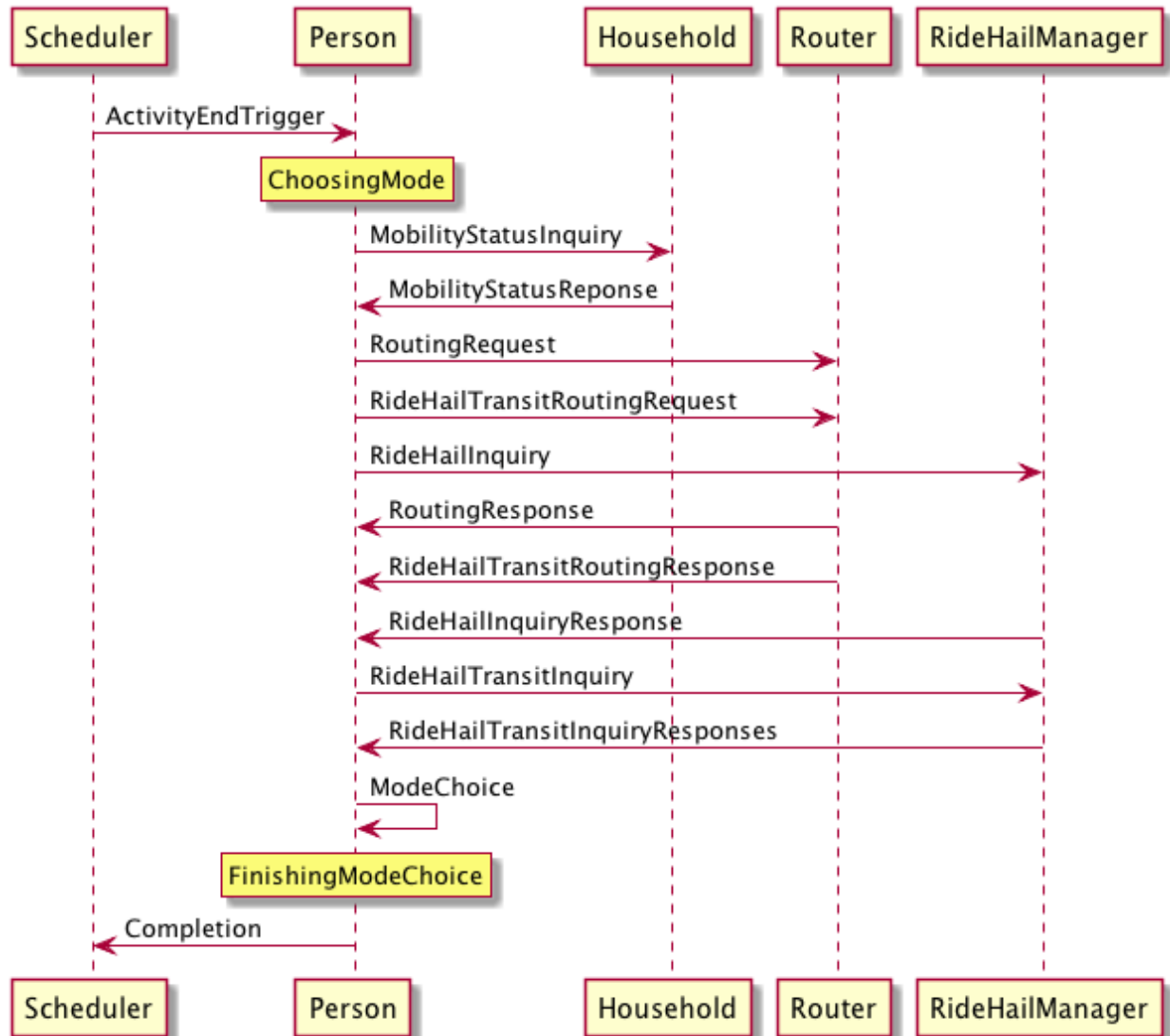
- Vector of link Ids
- Optional transit stop info (for any Transit leg, the boarding and alighting stop)
- A trajectory resolver which is responsible for translating the linkIds into coordinates

BeamTransitSegments contain:

- Origin stop
- Destination stop

9.1.2 ChoosesMode

The mode choice protocol involves gathering information, making a choice, confirming reservations if necessary, and then adapting if the chosen trip cannot be executed.



Gathering Information

1. The Person receives the `BeginModeChoiceTrigger` from the scheduler.
2. Person sends a `MobilityStatusInquiry` to thier Household.
3. Household returns a `MobilityStatusResponse`. Based on this response, the person optionally includes vehicles in the `ReservationRequest` sent to the Router.
4. Person sends a `ReservationRequest` to the Router.
5. Person sends a `RideHailInquiry` to the RideHailManager.
6. Person schedules a `FinalizeModeChoiceTrigger` to occur in the future (respresenting non-zero time to make a choice).
7. Person stays in `ChoosingMode` state until all results are recieved: `RoutingResponse`, `RideHailingInquiryResponse`, `FinalizeModeChoiceTrigger`. With each response, the data is stored locally for use in the mode choice.

Choosing and Reserving

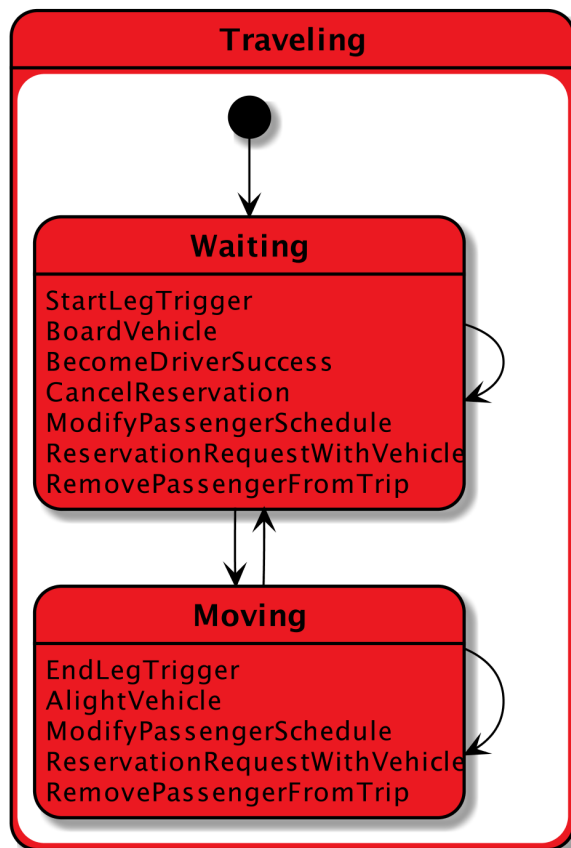
1. The Person evaluates the `ModeChoiceCalculator` which returns a chosen itinerary (in the form of an Emboided-BeamTrip) from the list of possible alternatives.

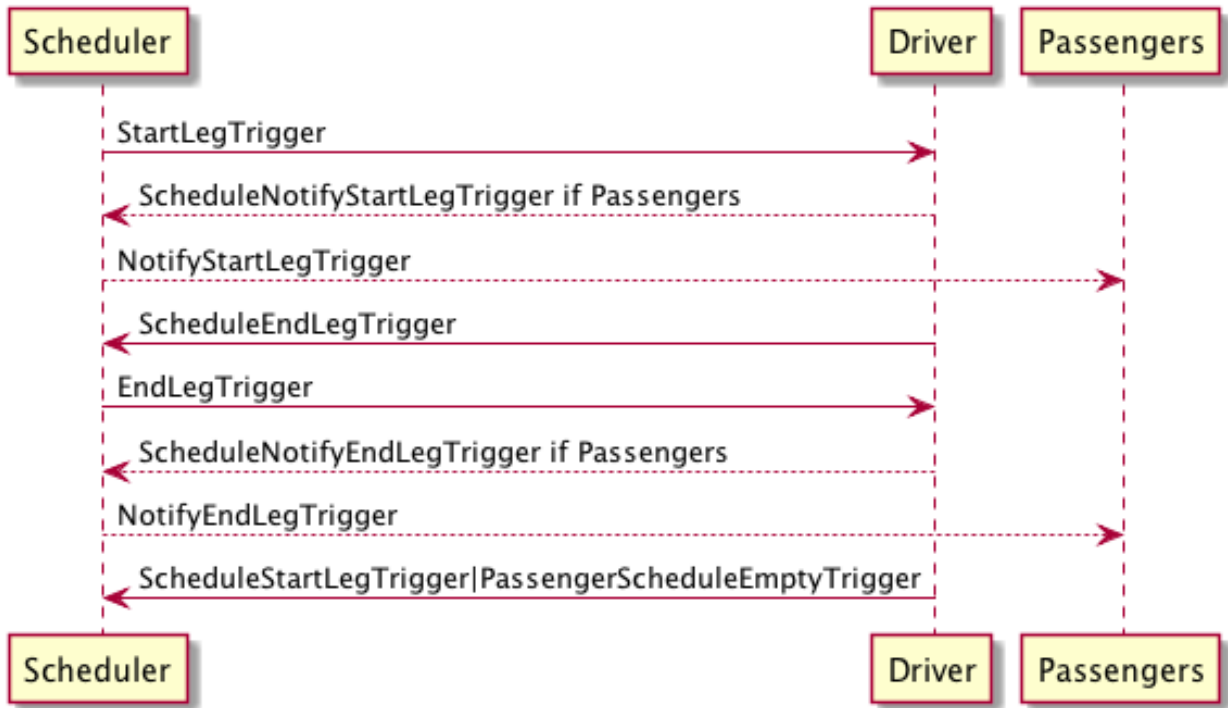
2. If a reservation is required to accomplish the chosen itinerary, the person sends `ReservationRequest` to all drivers in the itinerary (in the case of a transit trip) or a `ReserveRide` message to the `RideHailManager` in the case of a ride hail trip).
3. If reservation requests were sent, the Person waits (still in `ChoosingMode` state) for all responses to be returned. If any response is negative, the Person removes the chosen itinerary from their choice set, sends `RemovePassengerFromTrip` message to all drivers in the trip (if transit) and begins the mode choice process anew.
4. If all reservation responses are received and positive or if the trip does not require reservations at all, the person releases any reserved personal vehicles by sending `ReleaseVehicleReservation` messages to the Household and a `ResourceIsAvailableNotification` to themselves, the Person throws a `PersonDepartureEvent` and finally schedules a `PersonDepartureTrigger` to occur at the departure time of the trip.

9.2 Traveling

When a `PersonAgent` travels, she may transition from being a driver of a vehicle to being a passenger of a vehicle. The protocol for being a driver of a vehicle is listed separately below because that logic is implemented in its own trait (`DrivesVehicle`) to allow `BeamAgents` other than `PersonAgents` to drive vehicles. Some agents (e.g. `TransitDrivers`) may not be `Persons` and therefore do not “travel” but they do of course operate a vehicle and move around with it.

9.2.1 Driver





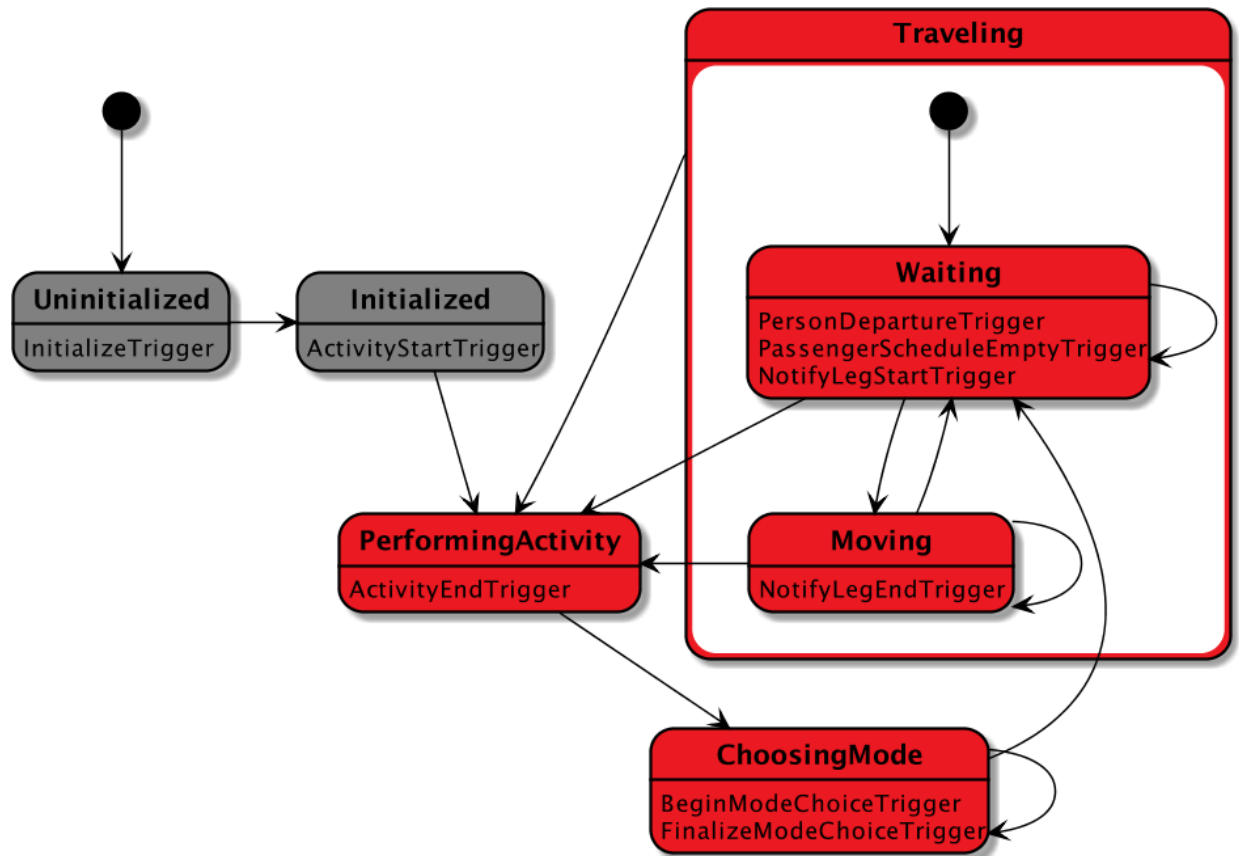
Starting Leg

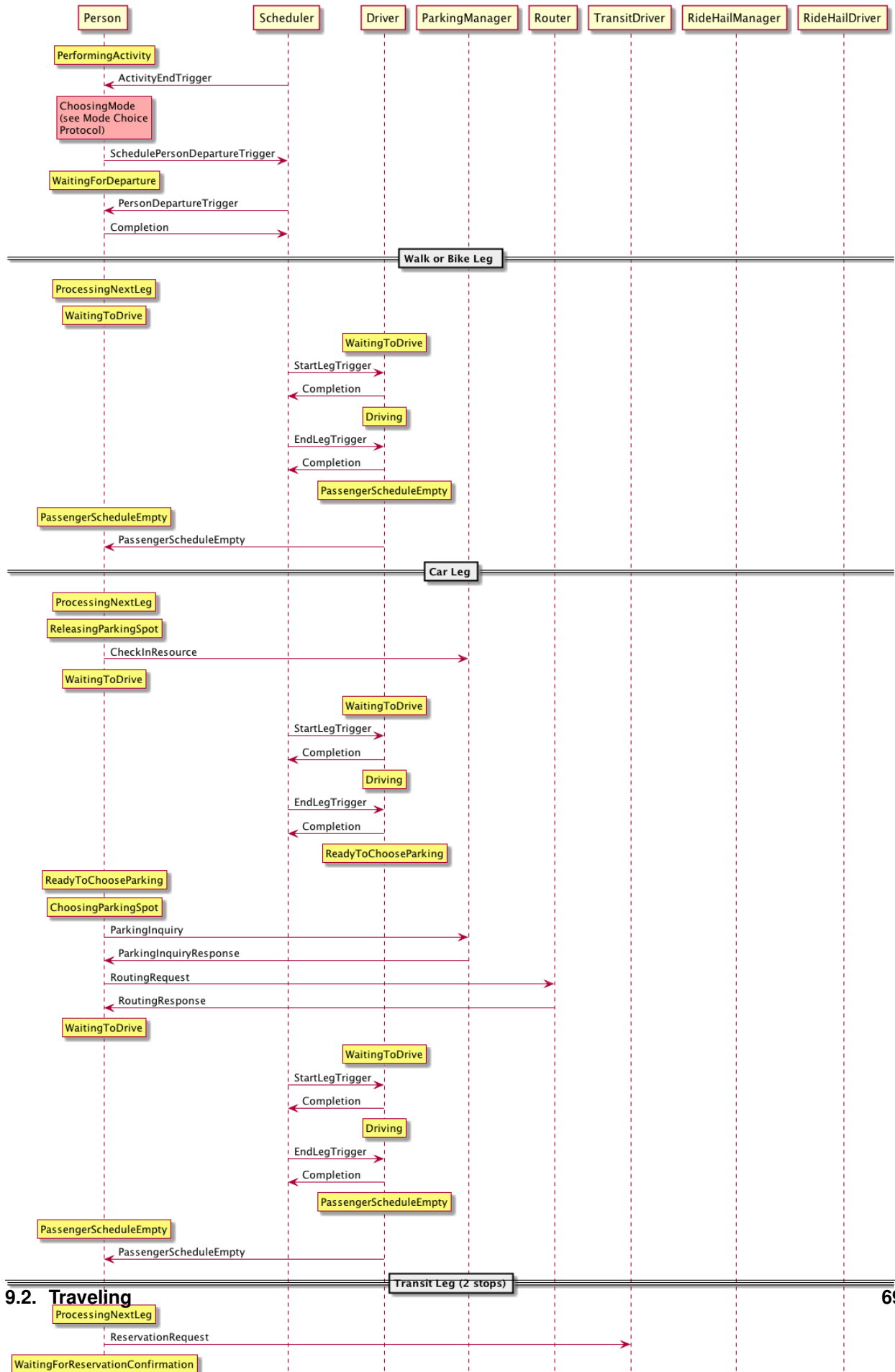
1. The Driver receives a StartLegTrigger from the Waiting state.
2. The Driver schedules NotifyLegStartTriggers for each rider in the PassengerSchedule associated with the current BeamLeg.
3. The Driver creates a list of borders from the PassengerSchedule associated with the BeamLeg to track which agents have yet to board the vehicle.
4. When all expected BoardVehicle messages are recieved by the Driver, the Driver schedules an EndLegTrigger and transitions to the Moving state.

Ending Leg

1. The Driver receives an EndLegTrigger from the Moving state.
2. The Driver schedules NotifyLegEndTriggers for all riders in the PassengerSchedule associated with the current BeamLeg.
3. The Driver creates a list of alighters from the PassengerSchedule associated with the BeamLeg to track which agents have yet to alight the vehicle.
4. When all expected AlightingConfirmation messages are recieved from the vehicle, the Driver publishes a Path-TraversalEvent and proceeds with the following steps.
5. If the Driver has more legs in the PassengerSchedule, she schedules a StartLegTrigger based on the start time of that BeamLeg.
6. Else the Driver schedules a PassengerScheduleEmptyTrigger to execute in the current tick.
7. The Driver transitions to the Waiting state.

9.2.2 Traveler





Starting Trip

1. The PersonAgent receives a PersonDepartureTrigger from the scheduler while in Waiting state. She executes the ProcessNextLeg Method described below.

ProcessNextLeg Method

The following protocol is used more than once by the traveler so it is defined here as a function with no arguments.

1. The Person checks the value of `_currentEmbodiedLeg` to see if `unbecomeDriverOnCompletion` is set to `TRUE`, if so, then the Person sends an `UnbecomeDriver` message to her vehicle and updates her `_currentVehicle` accordingly.
2. If there are no more legs in the `EmbodiedBeamTrip`, the PersonAgent either schedules the `ActivityEndTrigger` and transitions to the `PerformingActivity` state or, if there are no remaining activities in the person's plan, she transitions to the `Finished` state and schedules no further triggers.
3. If there are more legs in the `EmbodiedBeamTrip`, the PersonAgent processes the next leg in the trip. If `asDriver` for the next leg is `FALSE`, then the Person transitions to `Waiting` state and does nothing further.
4. If `asDriver` is true for the next leg, the Person creates a temporary passenger schedule for the next leg and sends it along with a `BecomeDriver` or a `ModifyPassengerSchedule` message, depending on whether this person is already the driver of the vehicle or if becoming the driver for the first time.
5. The person stays in the current state (which could be `Waiting` or `Moving` depending on the circumstances).

Driving Mission Completed

1. The PersonAgent receives a `PassengerScheduleEmptyTrigger` from the scheduler which indicates that as a driver, this Person has finished all legs in her `PassengerSchedule`.
2. The PersonAgent executes the `ProcessNextLeg` method.

Notify Start Leg

1. The PersonAgent receives a `NotifyLegStartTrigger`.
2. If the private field `_currentEmbodiedLeg` is non-empty or if the leg referred to in the trigger does not match the Person's next leg or if the Person's next leg has `asDriver` set to `TRUE`, this Person has received the `NotifyLegStartTrigger` too early, so she reschedules the `NotifyLegStartTrigger` to occur in the current tick, allowing other messages in her Actor mailbox to be processed first.
3. Otherwise, the PersonAgent sends an `BoardVehicle` message to the driver contained in the `EmbodiedBeamLeg` unless she is already a passenger in that vehicle.
4. The PersonAgent transitions to the `Moving` state.

Notify End Leg

1. The PersonAgent receives a `NotifyLegEndTrigger`.
2. If the private field `_currentEmbodiedLeg` is empty or the `currentBeamLeg` does not match the leg associated with the Trigger, this Person has received the `NotifyLegEndTrigger` too early, so she reschedules the `NotifyLegEndTrigger` to occur in the current tick, allowing other messages in her Actor mailbox to be processed first.
3. If another `EmbodiedBeamLeg` exists in her `EmbodiedBeamTrip` AND the `BeamVehicle` associated with the next `EmbodiedBeamTrip` is identical to the current `BeamVehicle`, then she does nothing other than update her internal state to note the end of the leg and transition to `Waiting`.
4. Else she sends the current driver an `AlightVehicle` message and executes the `ProcessNextLegModule` method.

9.3 Household

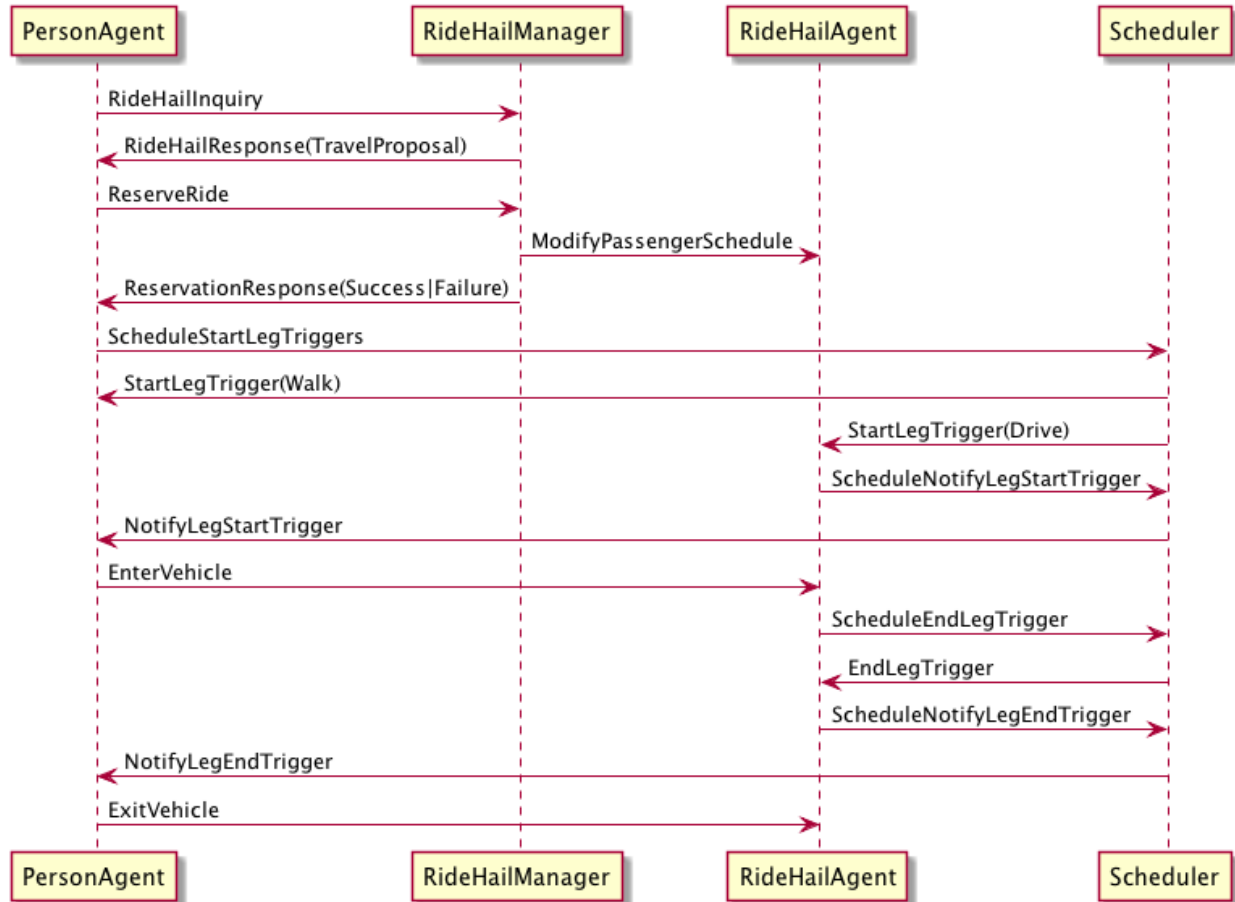
During initialization, we execute the rank and escort heuristic. Escorts and household vehicles are assigned to members.

1. The PersonAgent retrieves mobility status from her Household using a MobilityStatusInquiry message.
2. Household returns a MobilityStatusResponse message which notifies the person about two topics: a) whether she is an escortee (e.g. a child), an escorter (e.g. a parent), or traveling alone; b) the Id and location of at most one Car and at most one Bike that the person may use for their tour.
3. If the PersonAgent is an escortee, then she will enter a waiting state until she receives a AssignTrip message from her escorter which contains the BeamTrip that she will follow, at which point she schedules a PersonDepartureTrigger.
4. Else the PersonAgent goes through the mode choice process. After choosing a BeamTrip, she sends an appropriate BeamTrip to her escortees using the AssignTrip message.
5. The PersonAgent sends a VehicleConfirmationNotice to the Household, confirming whether or not she is using the Car or Bike. The Household will use this information to offer unused vehicles as options to subsequent household members.

9.3.1 Escort

9.4 RideHailing

The process of hailing a ride from a TNC is modeled after the real-world experience:



1. The PersonAgent inquires about the availability and pricing of the service using a RideHailingInquiry message.
2. The RideHailingManager responds with a RideHailingInquiryResponse.
3. The PersonAgent may choose to use the ride hailing service in the mode choice process.
4. The PersonAgent sends a ReserveRide message attempting to book the service.
5. The RideHailingManager responds with a ReservationResponse which either confirms the reservation or notifies that the resource is unavailable.

9.4.1 Inquiry

The RideHailingInquiry message contains:

- inquiryId
- customerId
- pickUpLocation
- departAt time
- destinationLocation

The RideHailingInquiryResponse message contains:

- inquiryId
- a Vector of TravelProposals

- an optional ReservationError

Each TravelProposal contains:

- RideHailingAgentLocation
- Time to Customer
- estimatedPrice
- estimatedTravelTime
- Route to customer
- Route from origin to destination

9.4.2 Reserve

The ReserveRide message contains:

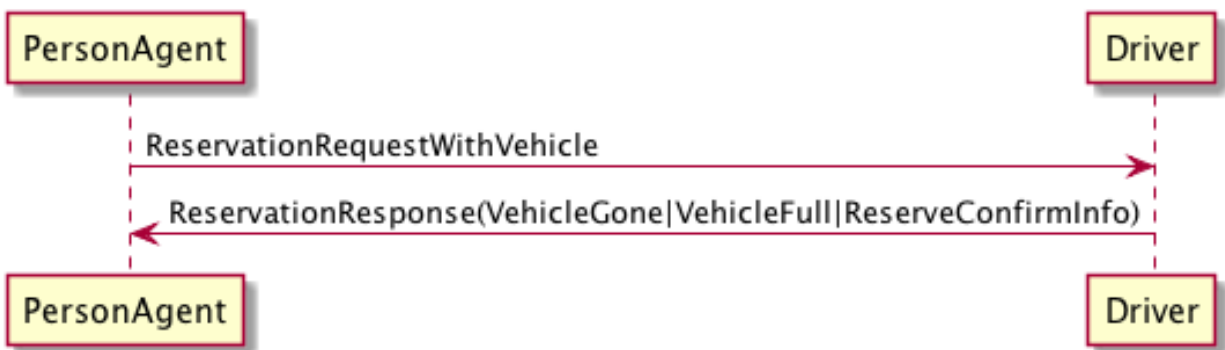
- inquiryId
- customerId in the form of a VehiclePersonId
- pickUpLocation
- departAt time
- destinationLocation)

The ReservationResponse message contains the request Id and either a ReservationError or if the reservation is successful, a ReserveConfirmInfo object with the following:

- DepartFrom BeamLeg.
- ArriveTo BeamLeg.
- PassengerVehicleId object containin the passenger and vehicle Ids.
- Vector of triggers to schedule.

9.5 Transit

Transit itineraries are returned by the router in the Trip Planning Protocol. In order to follow one of these itineraries, the PersonAgent must reserve a spot on the transit vehicle according to the following protocol:



1. PersonAgent sends ReservationRequest to the Driver.

2. The BeamVehicle forwards the reservation request to the Driver of the vehicle. The driver is responsible for managing the schedule and accepting/rejecting reservations from customers.
3. The Driver sends a ReservationConfirmation directly to the PersonAgent.
4. When the BeamVehicle makes it to the confirmed stop for boarding, the Driver sends a BoardingNotice to the PersonAgent.
5. The PersonAgent sends an BoardVehicle message to the Driver. 7. Also, concurrently, when the BeamVehicle is at the stop, the Driver sends an AlightingNotice to all passengers registered to alight at that stop. 8. Notified passengers send an AlightVehicle message to the Driver.

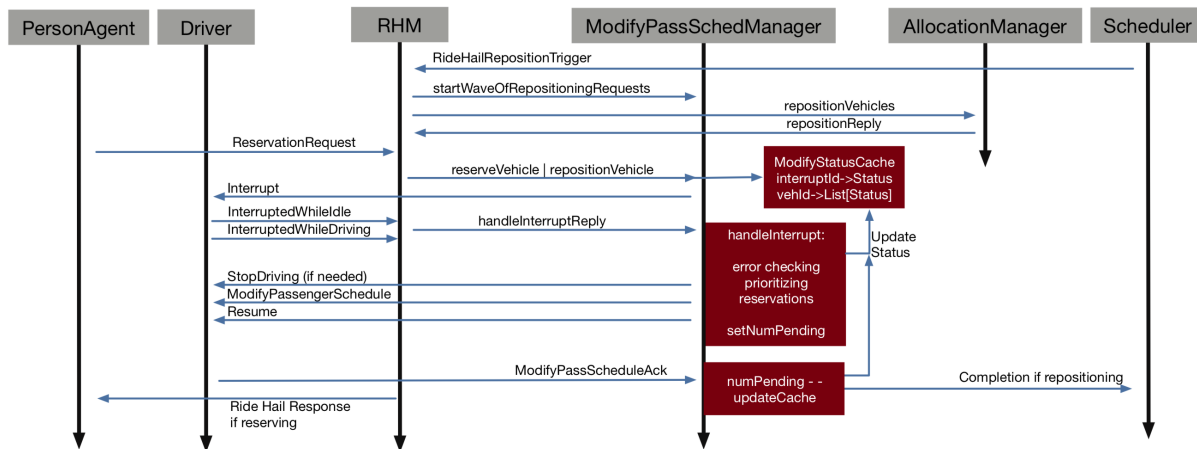
Because the reservation process ensures that vehicles will not exceed capacity, the Driver need not send an acknowledgement to the PersonAgent.

9.6 Refueling

???

9.7 Modify Passenger Schedule Manager

This protocol is deep into the weeds of the Ride Hail Manager but important for understanding how reservations and reposition requests are managed.



10.1 Git LFS

10.1.1 Setup git-lfs Server

1. From the AWS Management Console, launch the Amazon EC2 instance from an Amazon Machine Image (AMI) that has Ubuntu 64-bit as base operating system.
2. Choose a security group that will allow SSH access as well as port 8080 to access your git lfs server. You should only enable ingress from the IP addresses you wish to allow access to your server.
3. On AWS Management Console go to Services menu from top bar and open the Amazon S3 console.
4. Click Create Bucket, it will opens a new dialog window.
6. On the Name and region tab, provide appropriate name (should be DNS compliant) and desired region, then Click Next button in the bottom.
7. Leave Set properties as is and click Next again.
8. On Set Permissions tab, set read and write access for your git-lfs user. and Click next.
9. Verify your settings on Review tab. If you want to change something, choose Edit. If your current settings are correct, choose Create bucket.
10. Connect to the ec2 instance via SSH.
11. Add NodeSource APT repository for Debian-based distributions repository AND the PGP key for verifying packages:

```
$ curl -sL https://deb.nodesource.com/setup\_6.x | sudo -E bash -
```
12. Install Node.js from the Debian-based distributions repository:

```
$ sudo apt-get install -y nodejs
```
13. To confirm that Node.js was successfully installed on your system, you can run the following command:

```
$ node -v
```

If Node is installed, this command should print out something like this:

```
v6.9.1
```

14. To get the most up-to-date npm, you can run the command:

```
$ sudo npm install npm -global
```

15. Next, you can directly install git-lfs server using node package manager by executing following command:

```
$ sudo npm install node-git-lfs
```

16. Git LFS server offers two method of configuration, via environment variable or configuration file. At this step you have to define some environment variables to configure the server:

- LFS_BASE_URL - URL of the LFS server - **required**
- LFS_PORT - HTTP portal of the LFS server, default to 3000 - **required**
- LFS_STORE_TYPE - Object store type, can be either s3 (for AWS S3) or grid (for MongoDB GridFS), default to s3 - **required**
- LFS_AUTHENTICATOR_TYPE - Authenticator type, can be basic (for basic username and password), none (for no authentication), default to none - **required**
- LFS_AUTHENTICATOR_USERNAME - Username - **required**
- LFS_AUTHENTICATOR_PASSWORD - Password - **required**
- AWS_ACCESS_KEY - AWS access key - **required**
- AWS_SECRET_KEY - AWS secret key - **required**
- LFS_STORE_S3_BUCKET - AWS S3 bucket - **required**
- LFS_STORE_S3_ENDPOINT - AWS S3 endpoint, normally this will be set by region
- LFS_STORE_S3_REGION - AWS S3 region

Set Aws access key, secret ky and s3 details based on previous steps.

17. Now start git lfs server:

```
$ node-git-lfs
```

18. At the end, create file named .lfsconfig in you repository with following contents, update host and port based on your environment.

```
[lfs] url = "http://host:port/LBNL-UCB-STI/beam.git" batch = true access = basic
```

This will setup everything you need to setup and install a custom gitl-lfs server on Amazon instance and github repository will start pointing to the your custom server. There is no special installation or requirement for the clint, only thing that you need is to provide lfs user name and password on you client when you pull your contents for the first time.

10.2 Jenkins

10.2.1 Setup Jenkins Server

1. From the AWS Management Console, launch the Amazon EC2 instance from an Amazon Machine Image (AMI) that has Ubuntu 64-bit as base operating system.
2. Choose a security group that will allow SSH access as well as port 8080, 80 and 443 to access your Jenkins dashboard. You should only enable ingress from the IP addresses you wish to allow access to your server.

3. Connect to the instance via SSH.

4. Add oracle java apt repository:

```
$ sudo add-apt-repository ppa:webupd8team/java
```

5. Run commands to update system package index and install Java installer script:

```
$ sudo apt update; sudo apt install oracle-java8-installer
```

6. Add the repository key to the system:

```
$ wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key ↪add - .
```

7. Append the Debian package repository address to the server's sources:

```
$ echo deb https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/  
↪sources.list.d/jenkins.list
```

8. Run update so that apt-get will use the new repository:

```
$ sudo apt-get update
```

9. Install Jenkins and its dependencies, including Java:

```
$ sudo apt-get install jenkins
```

10. Start Jenkins:

```
$ sudo service jenkins start
```

11. Verify that it started successfully:

```
$ sudo service jenkins status
```

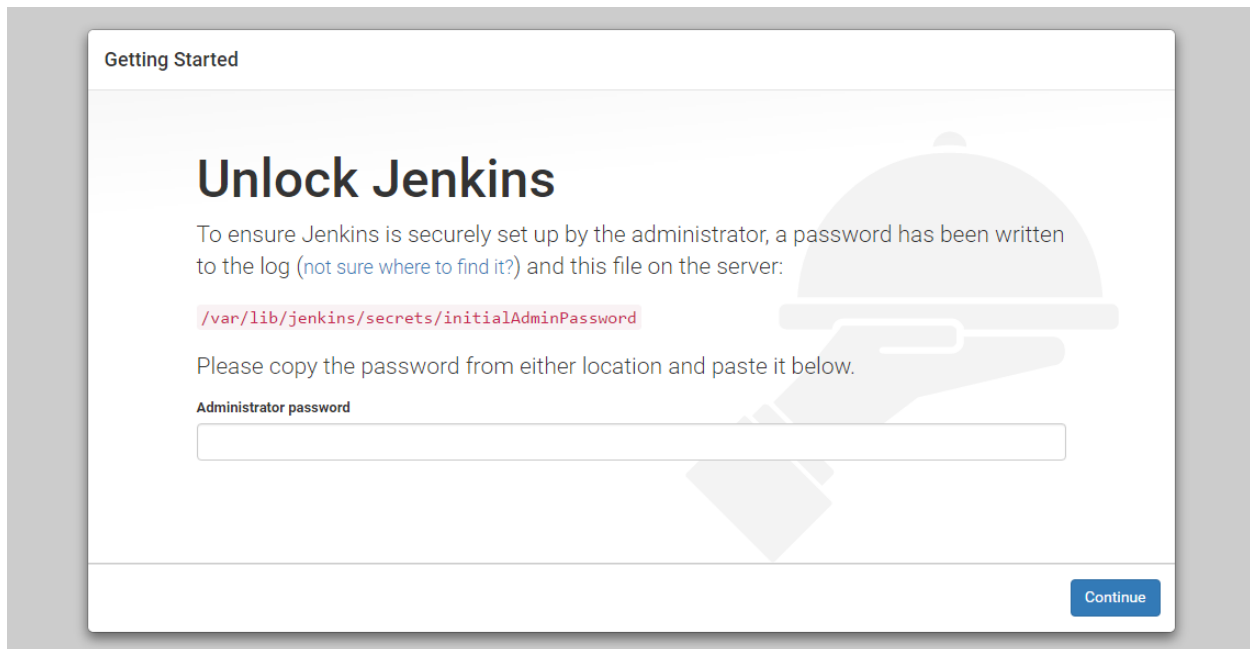
12. If everything went well, the beginning of the output should show that the service is active and configured to start at boot:

```
jenkins.service - LSB: Start Jenkins at boot time Loaded: loaded (/etc/init.d/jenkins; bad; vendor preset:  
enabled) Active:active (exited) since Thu 2017-04-20 16:51:13 UTC; 2min 7s ago Docs: man:systemd-  
sysv-generator(8)
```

13. To set up installation, visit Jenkins on its default port, 8080, using the server domain name or IP address:

http://ip_address_of_ec2_instance:8080

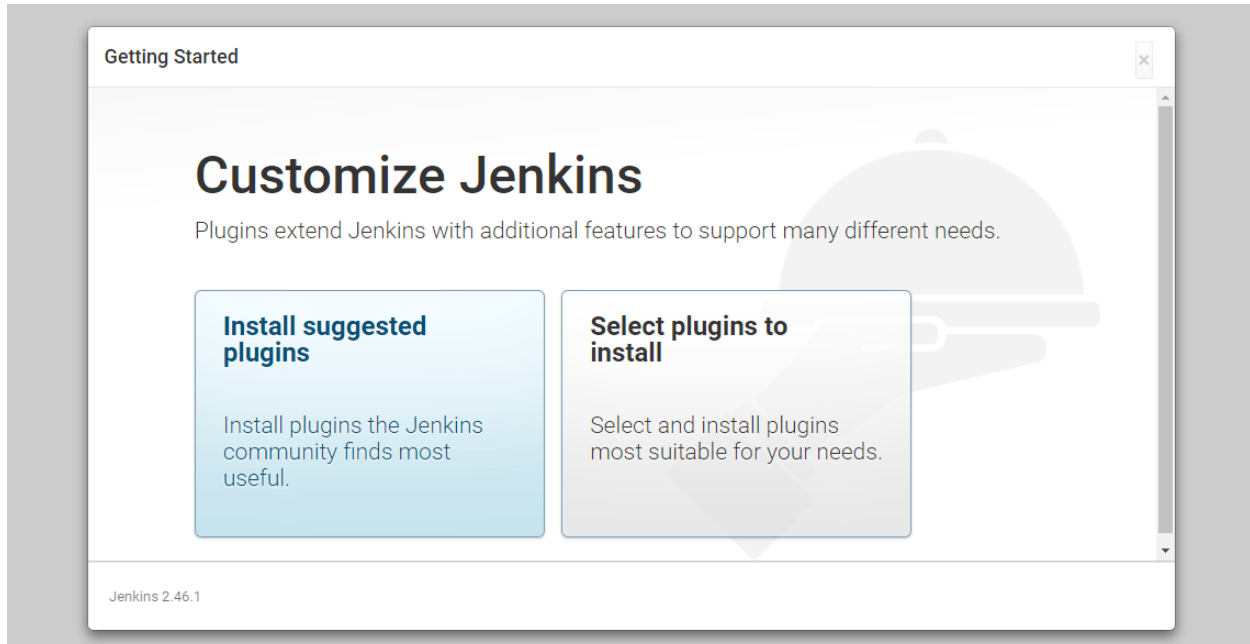
An “Unlock Jenkins” screen would appear, which displays the location of the initial password



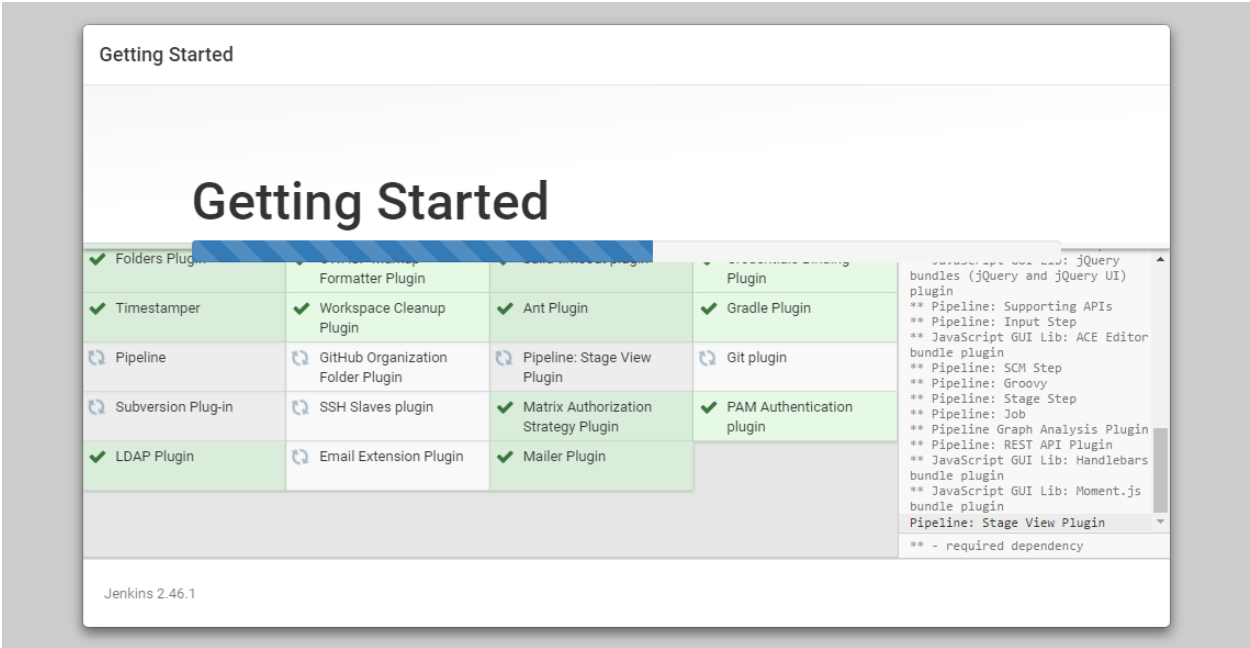
14. In the terminal window, use the cat command to display the password:

```
$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

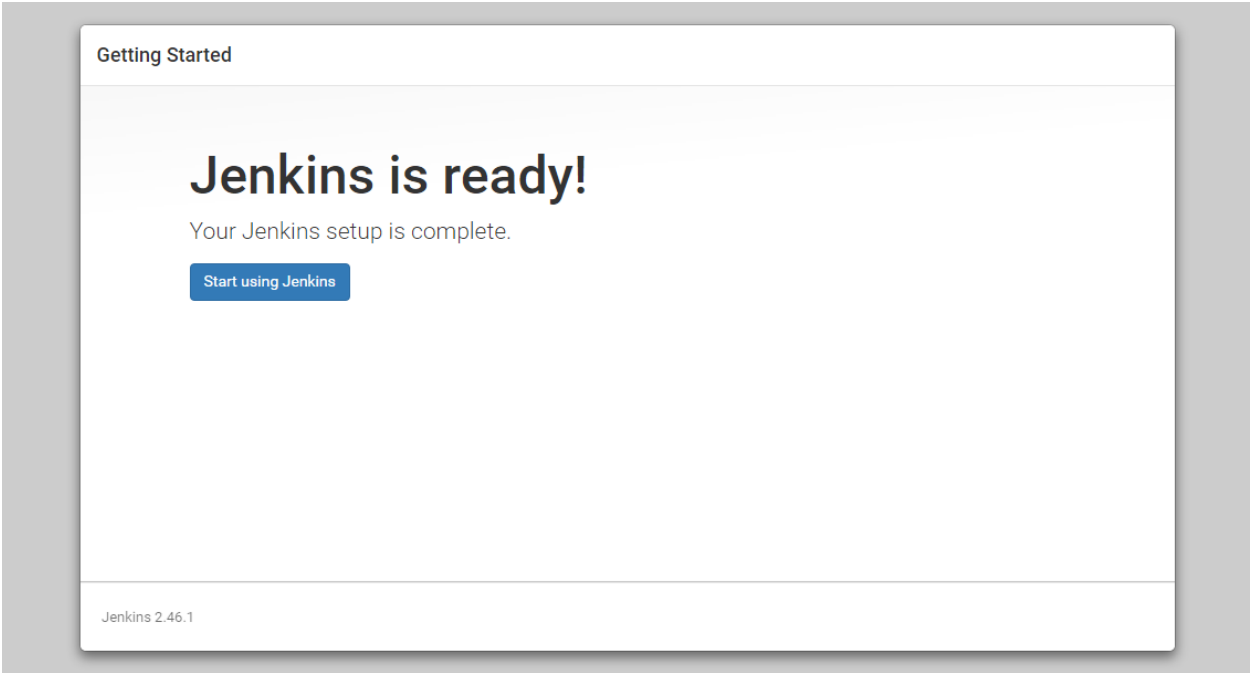
15. Copy the 32-character alphanumeric password from the terminal and paste it into the “Administrator password” field, then click “Continue”.



16. Click the “Install suggested plugins” option, which will immediately begin the installation process.



17. When the installation is complete, it prompt to set up the first administrative user. It's possible to skip this step and continue as admin using the initial password used above, but its batter to take a moment to create the user.



18. Once the first admin user is in place, you should see a “Jenkins is ready!” confirmation screen.

Getting Started

Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

Jenkins 2.46.1

Continue as admin

Save and Finish

19. Click “Start using Jenkins” to visit the main Jenkins dashboard.

search

Sammy Shark | log out

Jenkins

ENABLE AUTO REFRESH

New Item

People

Build History

Manage Jenkins

My Views

Credentials

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

Welcome to Jenkins!

Please [create new jobs](#) to get started.

add description

Page generated: Apr 20, 2017 7:41:49 PM UTC

[REST API](#)
[Jenkins ver. 2.46.1](#)

At this point, Jenkins has been successfully installed.

20. Update your package lists and install Nginx:

```
$ sudo apt-get install nginx
```

21. To check successful installation run:

```
$ nginx -v
```

22. Move into the proper directory where you want to put your certificates:

```
$ cd /etc/nginx
```

23. Generate a certificate:

```
$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/
↪cert.key -out /etc/nginx/cert.crt
```

24. Next you will need to edit the default Nginx configuration file:

```
$ sudo vi /etc/nginx/sites-enabled/default
```

25. Update the file with following contents:

```
server { listen 80; return 301 https://protect\T1\textdollarhost\protect\T1\textdollarrequest_uri;
}

server { listen 443; server_name beam-ci.tk;

    ssl_certificate /etc/nginx/cert.crt; ssl_certificate_key /etc/nginx/cert.key;

    ssl on; ssl_session_cache builtin:1000 shared:SSL:10m; ssl_protocols TLSv1 TLSv1.1
    TLSv1.2; ssl_ciphers HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
    ssl_prefer_server_ciphers on;

    access_log /var/log/nginx/jenkins.access.log;

    location / { proxy_set_header Host $host; proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; proxy_set_header X-
        Forwarded-Proto $scheme;

        # Fix the "It appears that your reverse proxy set up is broken" error. proxy_pass http://localhost:
        8080; proxy_read_timeout 90;

        proxy_redirect http://localhost:8080 https://beam-ci.tk;
    }
}
```

26. For Jenkins to work with Nginx, you need to update the Jenkins config to listen only on the localhost interface instead of all (0.0.0.0), to ensure traffic gets handled properly. This is an important step because if Jenkins is still listening on all interfaces, then it will still potentially be accessible via its original port (8080).

27. Modify the /etc/default/jenkins configuration file to make these adjustments:

```
$ sudo vi /etc/default/jenkins
```

28. Locate the JENKINS_ARGS line and update it to look like the following:

```
$ JENKINS_ARGS="--webroot=/var/cache/$NAME/war --httpListenAddress=127.0.0.1 --
↪httpPort=$HTTP_PORT -ajp13Port=$AJP_PORT"
```

29. Then go ahead and restart Jenkins:

```
$ sudo service jenkins restart
```

30. After that restart Nginx:

```
$ sudo service nginx restart
```

You should now be able to visit your domain using either HTTP or HTTPS, and the Jenkins site will be served securely. You will see a certificate warning because you used a self-signed certificate.

31. Next you install certbot to setup nginx with as CA certificate. Certbot team maintains a PPA. Once you add it to your list of repositories all you'll need to do is apt-get the following packages:

```
$ sudo add-apt-repository ppa:certbot/certbot
```

32. Run apt update:

```
$ sudo apt-get update
```

33. Install certbot for Nginx:

```
$ sudo apt-get install python-certbot-nginx
```

34. Get a certificate and have Certbot edit Nginx configuration automatically, run the following command:

```
$ sudo certbot -nginx
```

35. The Certbot packages on your system come with a cron job that will renew your certificates automatically before they expire. Since Let's Encrypt certificates last for 90 days, it's highly advisable to take advantage of this feature. You can test automatic renewal for your certificates by running this command:

```
$ sudo certbot renew --dry-run
```

36. Restart Nginx:

```
$ sudo service nginx restart
```

37. Go to AWS management console and update the Security Group associated with jenkins server by removing the port 8080, that you added in step 2.

10.2.2 Setup Jenkins Slave

Now configure a Jenkins slave for pipeline configuration. You need the slave AMI to spawn automatic EC2 instance on new build jobs.

1. Create Amazon EC2 instance from an Amazon Machine Image (AMI) that has Ubuntu 64-bit as base operating system.
2. Choose a security group that will allow only SSH access to your master (and temporarily for your personal system).
3. Connect to the instance via SSH.
4. Add oracle java apt repository and git-lfs:

```
$ sudo add-apt-repository ppa:webupd8team/java*  
$ sudo curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.  
→deb.sh | sudo bash*
```

5. Run commands to update system package index:


```
$ sudo apt update
```

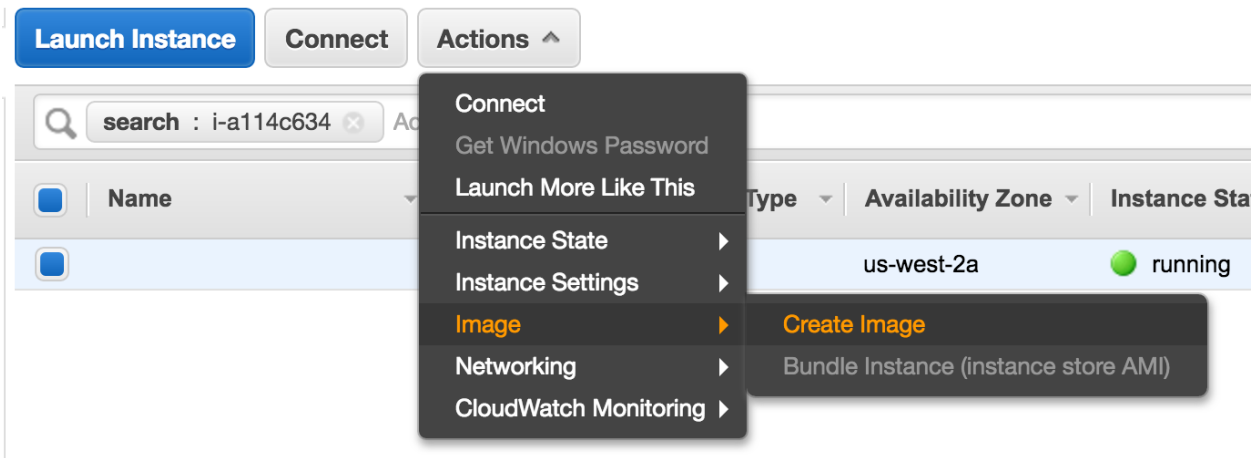
6. Install Java and other dependency components, there is no need to install any jenkins component or service. Jenkins automatically deploy an agent as it initiates the build:

```
$ sudo apt install git docker oracle-java8-installer git-lfs=2.3.4
```

7. SSH master that you created in last topic and from inside master again ssh your newly created slave, just to test the communication:

```
$ ssh ubuntu@<slave_ip_address>
```

8. In EC2 Instances pane, click on your Jenkins slave instance you just configure, and create a new image.



9. On Create Image dialog, name the image and select “Delete on Termination”. It makes slave instance disposable, if there are any build artifacts, job should save them, that will send them to your master.

Create Image

Instance ID ⓘ

i-a114c634

Image name ⓘ

Jenkins-slave AMI

Image description ⓘ

No reboot ⓘ

☐

Instance Volumes

| Volume Type ⓘ | Device ⓘ | Snapshot ⓘ | Size (GiB) ⓘ | Volume Type ⓘ | IOPS ⓘ | Throughput (MB/s) ⓘ | Delete on Termination ⓘ | Encrypted ⓘ |
|---------------|-----------|---------------|--------------|---------------------|------------|---------------------|-------------------------------------|---------------|
| Root | /dev/xvda | snap-d465048a | 30 | General Purpose SSD | 100 / 3000 | N/A | <input checked="" type="checkbox"/> | Not Encrypted |

Add New Volume

Total size of EBS Volumes: 30 GiB

When you create an EBS image, an EBS snapshot will also be created for each of the above volumes.

Cancel

Create Image

10. Once image creation process completes, just copy the AMI ID, you need it for master configuration.

| AMI Name | AMI ID |
|-------------------|--------------|
| Jenkins-slave AMI | ami-5fe9233f |

- Update the Slave security group and remove all other IP addresses except master. You should only enable ingress from the IP addresses you wish to allow access to your slave.

| Name | Group ID | Group Name | VPC ID | Description |
|------|----------|---------------|--------|--|
| | | jenkins-slave | | A security group for jenkins slave instances |

Security Group: sg-14b1c07f

Description Inbound Outbound Tags

Edit

| Type | Protocol | Port Range | Source | Description |
|------|----------|------------|-----------|-------------|
| SSH | TCP | 22 | 0.0.0.0/0 | |
| SSH | TCP | 22 | :::0 | |

- At the end drop slave instance, its not needed anymore.

10.2.3 Configure Jenkins Master

Now start configuring Jenkins master, so it can spawn new slave instance on demand.

- Once Master and Slave are setup, login to Jenkins server administrative console as admin.
- On the left-hand side, click Manage Jenkins, and then click Manage Plugins.
- Click on the Available tab, and then enter Amazon EC2 plugin at the top right.

Filter:

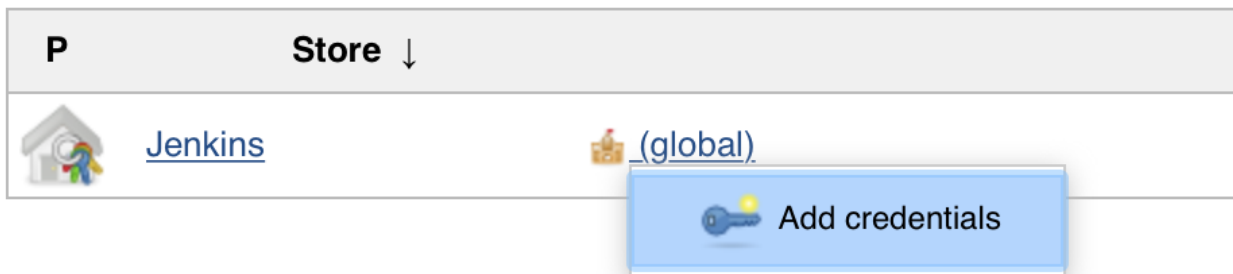
Updates Available Installed Advanced

| Install | Name | Version |
|-------------------------------------|--|---------|
| <input checked="" type="checkbox"/> | Amazon EC2 plugin Allow Jenkins to start slaves on EC2 or Eucalyptus on demand, and kill them as they get unused. | 1.29 |

Update information obtained: 1 day 7 hr ago

- Select the checkbox next to Amazon EC2 plugin, and then click Install without restart.
- Once the installation is done, click Go back to the top page. 4. On the sidebar, click on Credentials, hover (global) for finding the sub menu and add a credential.

Stores scoped to Jenkins



6. Choose AWS Credentials, and limit the scope to System, complete the form, if you make an error, Jenkins will add an error below the secret key. Jenkins uses access key ID and secret access key to interface with Amazon EC2.

Kind AWS Credentials

Scope System (Jenkins and nodes only) ?

Access Key ID [REDACTED]

Secret Access Key

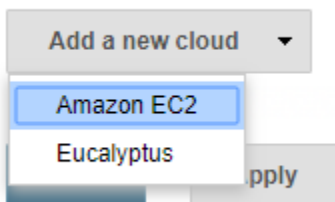
These credentials are valid and have access to 4 availability zones

ID AWS ?

Description AWS ?

OK

7. Click on Manage Jenkins, and then Configure System.
8. Scroll all the way down to the section that says Cloud.
9. Click Add a new cloud, and select Amazon EC2. A collection of new fields appears.




10. Select Amazon EC2 Credentials that you just created. EC2 Key Pair's Private key is a key generated when creating a new EC2 image on AWS.


Cloud


Amazon EC2

Name


Amazon EC2 Credentials 


AWS IAM Access Key used to connect to EC2. If not specified, implicit authentication mechanisms are used (IAM roles...)

Use EC2 instance profile to obtain credentials ☐ 

Region 


The regions will be populated once the keys above are entered.

EC2 Key Pair's Private Key 



11. Complete the form, choose a Region, Instance Type, label and set Idle termination time. If the slave becomes idle during this time, the instance will be terminated.

AMIs

Description 

AMI ID


Instance Type

EBS Optimized ☐


Availability Zone

☐ Use Spot Instance

Security group names

Remote FS root 


Remote user


AMI Type 


Root command prefix

Slave command prefix


Remote ssh port

Labels 

Usage 

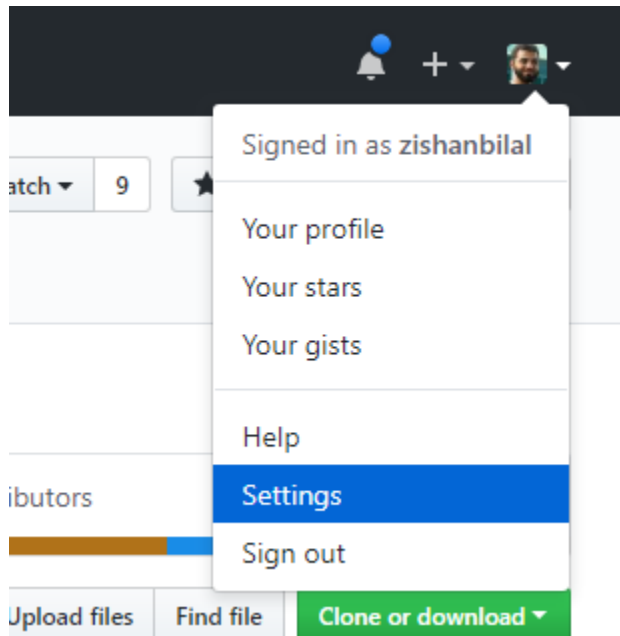
Idle termination time 

Init script



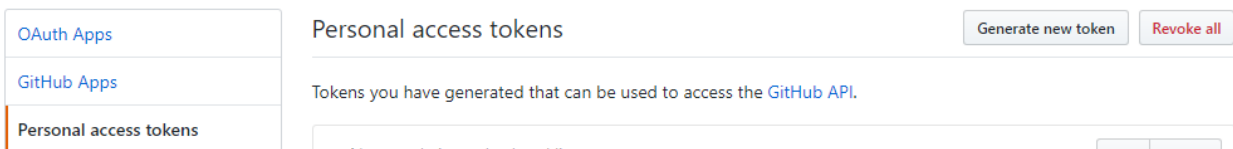
12. In order for Jenkins to watch GitHub projects, you will need to create a Personal Access Token in your GitHub account.

Now go to GitHub and signing into your account and click on user icon in the upper-right hand corner and select Settings from the drop down menu.



13. On Settings page, locate the Developer settings section on the left-hand menu and go to Personal access tokens and click on Generate new token button.

Settings / Developer settings



14. In the Token description box, add a description that will allow you to recognize it later.

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Token description

Jenkins integration for CI/CD

What's this token for?

15. In the Select scopes section, check the repo:status, repo:public_repo and admin:org_hook boxes. These will allow Jenkins to update commit statuses and to create webhooks for the project. If you are using a private repository, you will need to select the general repo permission instead of the repo sub items.

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

| | |
|---|--|
| <input type="checkbox"/> repo | Full control of private repositories |
| <input checked="" type="checkbox"/> repo:status | Access commit status |
| <input type="checkbox"/> repo_deployment | Access deployment status |
| <input checked="" type="checkbox"/> public_repo | Access public repositories |
| <input type="checkbox"/> admin:org | Full control of orgs and teams |
| <input type="checkbox"/> write:org | Read and write org and team membership |
| <input type="checkbox"/> read:org | Read org and team membership |
| <input type="checkbox"/> admin:public_key | Full control of user public keys |
| <input type="checkbox"/> write:public_key | Write user public keys |
| <input type="checkbox"/> read:public_key | Read user public keys |
| <input type="checkbox"/> admin:repo_hook | Full control of repository hooks |
| <input type="checkbox"/> write:repo_hook | Write repository hooks |
| <input type="checkbox"/> read:repo_hook | Read repository hooks |
| <input checked="" type="checkbox"/> admin:org_hook | Full control of organization hooks |
| <input type="checkbox"/> gist | Create gists |
| <input type="checkbox"/> notifications | Access notifications |

16. When you are finished, click Generate token at the bottom.


17. You will be redirected back to the Personal access tokens index page and your new token will displayed.

Personal access tokens

[Generate new token](#)[Revoke all](#)

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your new personal access token now. You won't be able to see it again!

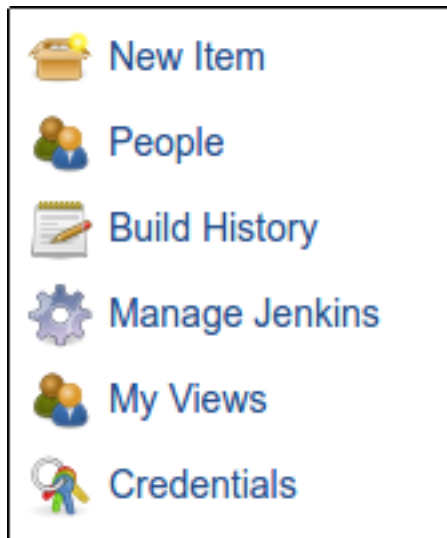
✓ 91b36366bca8c610e7312d95af87c4a823e0c177 

[Edit](#)[Delete](#)

18. Copy the token now so that you can reference it later.

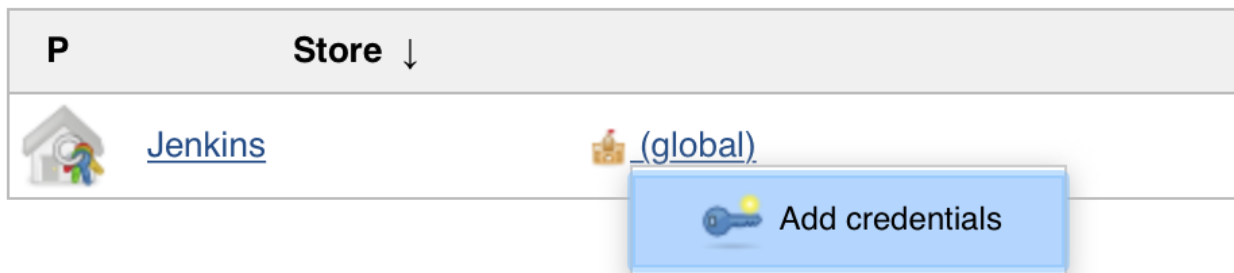
Now that you have a token, you need to add it to your Jenkins server so it can automatically set up webhooks. Log into your Jenkins web interface using the administrative account you configured during installation.

19. On Jenkins main dashboard, click Credentials in the left hand menu.



20. Click the arrow next to (global) within the Jenkins scope. In the box that appears, click Add credentials.

Stores scoped to Jenkins



21. From Kind drop down menu, select Secret text. In the Secret field, paste your GitHub personal access token. Fill out the Description field so that you will be able to identify this entry at a later date and press OK button in the bottom.

Kind **Secret text**

Scope **Global (Jenkins, nodes, items, all child items, etc)**

Secret **.....**

ID

Description **GitHub personal access token**

OK

22. Jenkins dashboard, click Manage Jenkins in the left hand menu and then click Configure System.

Manage Jenkins



[Configure System](#)

Configure global settings and paths.



[Configure Global Security](#)

Secure Jenkins; define who is allowed to access/use the system.

23. Find the section with title GitHub. Click the Add GitHub Server button and then select GitHub Server.

GitHub

GitHub Servers **Add GitHub Server**

GitHub Server

Advanced...

GitHub Enterprise Servers

Add

24. In the Credentials drop down menu, select your GitHub personal access token that you added in the last section.

GitHub Server

API URL **https://api.github.com**

Credentials **GitHub personal access token** **Add**

Credentials verified for user sammytheshark, rate limit: 4997

Test connection

Manage hooks ☒

25. Click the Test connection button. Jenkins will make a test API call to your account and verify connectivity. On successful connectivity click Save.

10.2.4 Configure Jenkins Jobs

Once Jenkins is installed on master and its configured with slave, cloud and github. The only thing we need now, before configuring the jobs, is to install a set of plugins.

1. On the left-hand side of Jenkins dashboard, click Manage Jenkins, and then click Manage Plugins.
2. Click on the Available tab, and then enter plugin name at the top right to install following set of plugins.
 - Gradle Plugin: This plugin allows Jenkins to invoke Gradle build scripts directly.
 - Build Timeout: This plugin allows builds to be automatically terminated after the specified amount of time has elapsed.
 - HTML5 Notifier Plugin: The HTML5 Notifier Plugin provides W3C Web Notifications support for builds.
 - Notification Plugin: you can notify on deploying, on master failure/back to normal, etc.
 - HTTP Request Plugin: This plugin sends a http request to a url with some parameters.
 - embeddable-build-status: Fancy but I love to have a status badge on my README
 - Timestamp: It adds time information in our build output.
 - AnsiColor: Because some tools (lint, test) output string with bash color and Jenkins do not render the color without it.
 - Green Balls: Because green is better than blue!
3. Back in the main Jenkins dashboard, click New Item in the left hand menu:
4. Enter a name for your new pipeline in the Enter an item name field. Afterwards, select Freestyle Project as the item type and Click the OK button at the bottom to move on.

Enter an item name

beam-ci

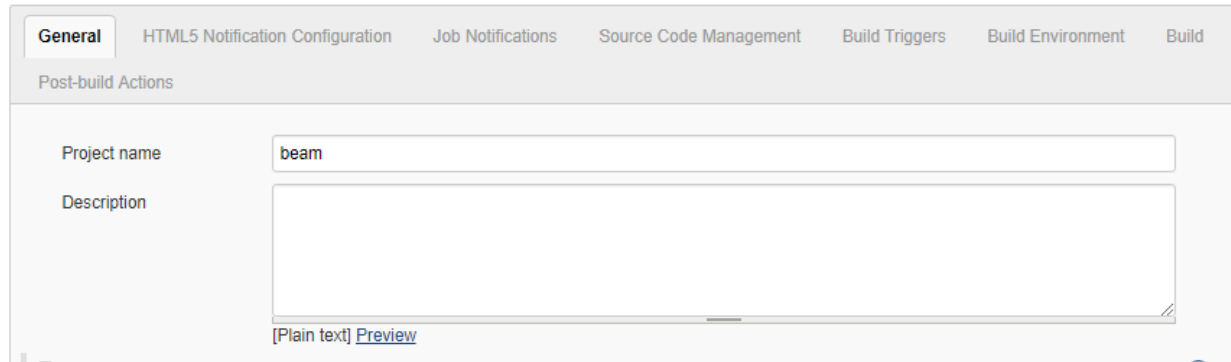
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

OK

5. On the next screen, specify Project name and description.



General HTML5 Notification Configuration Job Notifications Source Code Management Build Triggers Build Environment Build

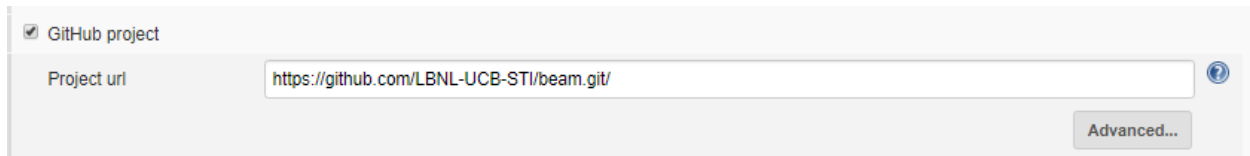
Post-build Actions

Project name beam

Description

[Plain text] [Preview](#)

6. Then check the GitHub project box. In the Project url field that appears, enter your project's GitHub repository URL.




☒ GitHub project

Project url <https://github.com/LBNL-UCB-STI/beam.git/> ⓘ

Advanced...

7. In the HTML5 Notification Configuration section left unchecked Skip HTML5 Notifications? Checkbox, to receive browser notifications against our builds



HTML5 Notification Configuration

Skip HTML5 Notifications? ☐ ⓘ

8. To configure Glip Notifications with Jenkins build you need to configure notification endpoint under Job Notification section. Select JSON in Format drop-down, HTML in Protocol and to obtain end point URL follow steps 8.1 through 8.3.

Job Notifications

Notification Endpoints

Format
JSON

Protocol
HTTP

Event
Job Finalized

Job lifecycle event triggering notification

URL Source
Plain Text

URL
https://hooks.glip.com/webhook/09154bbc-4e5d-4219-84af-65i

Where to send messages

Timeout
30000

Timeout (in ms)

Retries
0

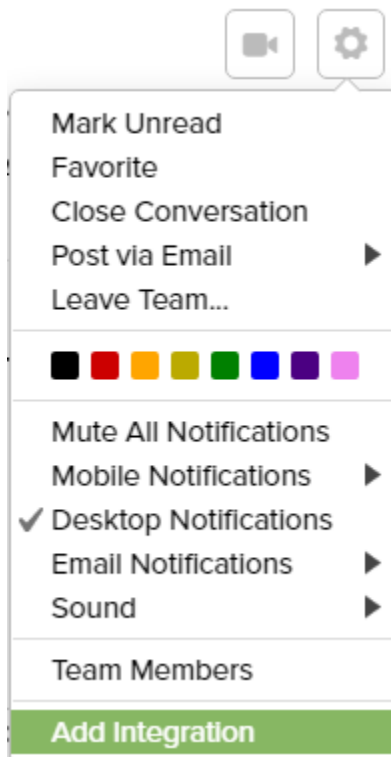
Retries

Log
0

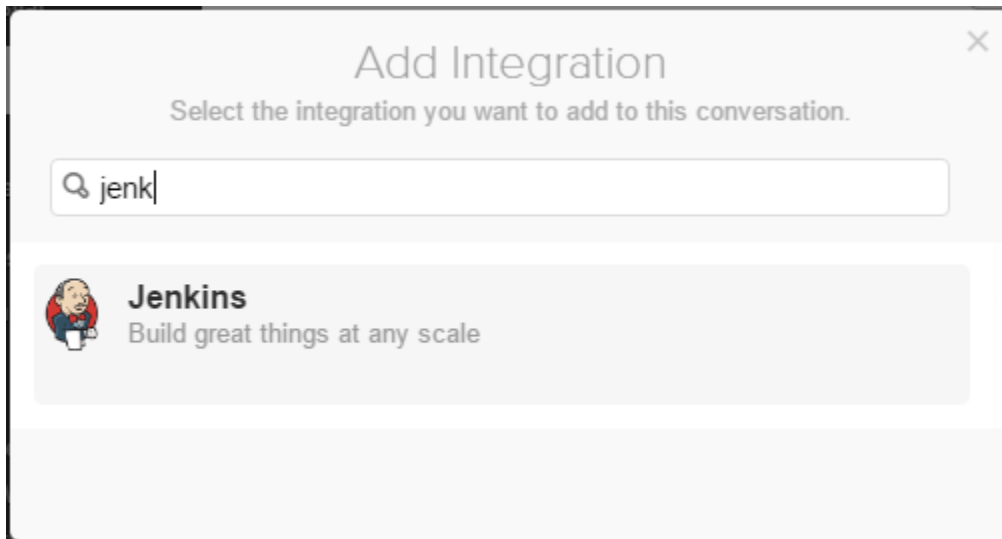
Number lines of log messages to send. Use -1 for all (use with caution).

Save
Apply

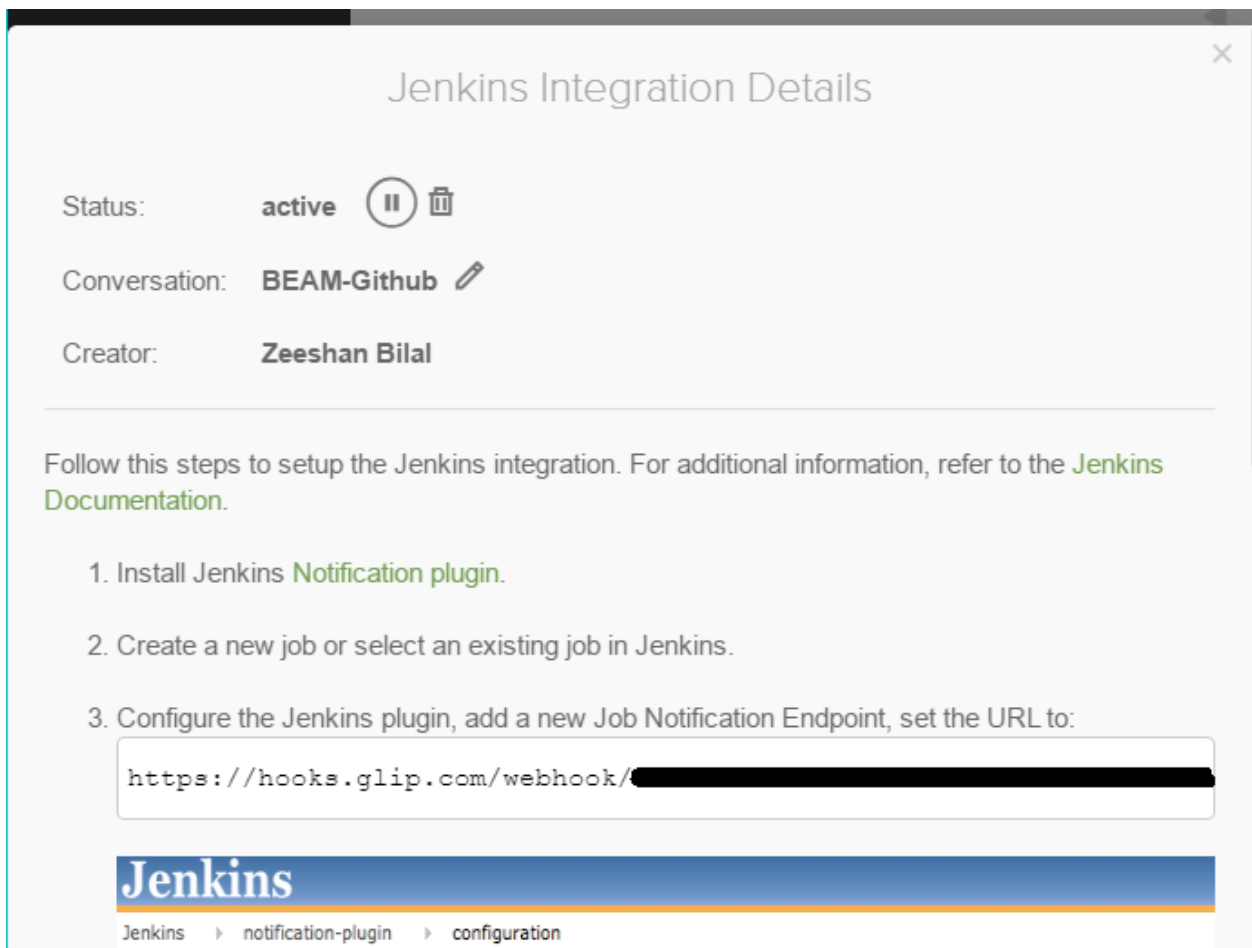
8.1. Open Glip and go to your desired channel where you want to receive notifications and then click top right button for Conversation Settings. It will open a menu, click Add Integration menu item.



8.2. On Add Integration dialog search Jenkins and click on the Jenkins Integration option.



8.3. A new window would appear with integration steps, copy the URL from this window and use in the above step.



9. At the end of notification section check Execute concurrent build if necessary and Restrict where this project can run and specify the label that we mentioned in last section while configuring master.

☒ Execute concurrent builds if necessary

☒ Restrict where this project can be run

Label Expression

[Label ec2](#) is serviced by 1 node and 1 cloud. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

Advanced...

10. In Source Code Management specify the beam github url against Repository URL and select appropriate credentials. Put ** for all branches, to activate build for all available bit hub branches.

Source Code Management

☐ None

☒ Git

Repositories

Repository URL

Credentials Add

Advanced...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

Add Branch

Repository browser

Additional Behaviours

☒ Git LFS pull after checkout

Add

11. Next, in the Build Triggers section, check the GitHub hook trigger for GITScm polling box.

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☐ Build after other projects are built

☐ Build periodically

☒ GitHub hook trigger for GITScm polling

☐ Poll SCM

12. Under Build Environment section, click Abort build if it's stuck and specify the timeout. Enable timestamps to Console output and select xterm in ANSI color option and in the end specify the build name pattern for more readable build names.

Build Environment

☐ Delete workspace before build starts
☐ Use secret text(s) or file(s) ?
☒ Abort the build if it's stuck

Time-out strategy: No Activity ?
 Timeout seconds: 3600 ?

Time-out variable:
Set a build timeout environment variable

Time-out actions:

Abort the build X ?

Add action

☒ Add timestamps to the Console Output
☒ Color ANSI Console Output

ANSI color map: xterm ?

☐ Generate Release Notes
☒ Set Build Name ?

Build Name: #{BUILD_NUMBER}-\${GIT_BRANCH} ?

Advanced...

☐ With Ant ?

13. Last but not least, in Build section add a gradle build step, check Use Gradle Wrapper and specify the gradle task for build.

Build

Invoke Gradle script X ?

☐ Invoke Gradle ?
☒ Use Gradle Wrapper ?

Make gradlew executable ☐

Wrapper location: ?

Tasks: build ?

Advanced...

Add build step

10.2.5 Configure Periodic Jobs

You can schedule any Jenkins job to run periodically based on provided schedule. To configure periodic build follow the steps below:

1. First click on Configure menu item from menu on left hand side of Job/Project home page.

2. On the next (configuration) page, go to *Build Triggers* section.

Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ GitHub hook trigger for GITScm polling
- ☐ Poll SCM

3. Click on check box labeled *Build periodically* to enable the option. It will expand and ask for Schedule with a warning message some thing like, No schedules so will never run.

Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☒ Build periodically

Schedule

 No schedules so will never run

4. You have to specify a schedule by following the similar syntax of cron job as a line consists of 5 fields separated by TAB or whitespace:

MINUTE HOUR DOM MONTH DOW

- MINUTE Minutes within the hour (0–59)
- HOUR The hour of the day (0–23)
- DOM The day of the month (1–31)
- MONTH The month (1–12)
- DOW The day of the week (0–7) where 0 and 7 are Sunday.

To schedule once daily every 24 hours for only 5 working days, we need to specify some thing like:

```
H 0 * * 1-5
```

Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☒ Build periodically

Schedule

H 0 * * 1-5

Would last have run at Friday, April 13, 2018 12:52:21 AM UTC; would next run at Monday, April 16, 2018 12:52:21 AM UTC.

As you specify the schedule, warning would be replaced with a descriptive schedule.

5. Save the configurations and now you have setup job to run periodically.

10.2.6 References

https://d0.awsstatic.com/whitepapers/DevOps/Jenkins_on_AWS.pdf

<https://www.digitalocean.com/community/tutorials/how-to-configure-nginx-with-ssl-as-a-reverse-proxy-for-jenkins>

<https://www.digitalocean.com/community/tutorials/how-to-set-up-continuous-integration-pipelines-in-jenkins-on-ubuntu-16-04>

<https://jmaitrehenry.ca/2016/08/04/how-to-install-a-jenkins-master-that-spawn-slaves-on-demand-with-aws-ec2>

10.3 Automated Cloud Deployment

10.3.1 Automatic Image (AMI) Update

In Automated Cloud Deployment capability, there is a baseline image (AMI) that used to instantiate new EC2 instance. It contains copy of git repository and gradle dependency libraries. All of these are outdated in few days due to active development of BEAM. And when we start instance from an outdated image it take additional time to update them before starting the simulation/run. This process help Cloud Automatic Deployment to keep up to date image for fast execution. To trigger this update process a Cloud Watch Event is setup with one week frequency. This event triggers an AWS Lambda (named *updateDependencies*) and lambda then starts an instance from the outdated image with instructions to update the image with latest LFS files for pre configured branches (these branches are mentioned in its environment variables that we can configure easily without any change in lambda code). One LFS files and gradle dependencies are updated in the new instance, the instance invoke a new lambda (named *updateBeamAMI*) to take its new image. This new lambda creates an image of the instance, terminate the instance and update this new image id to Automated Cloud Deployment process for future use.

This process is designed to get latest LFS files from different branches. To add a new branch or update existing one, an environment variable named *BRANCHES* need to update with space as branch name delimiter.

CHAPTER 11

Indices and tables

- `genindex`
- `modindex`
- `search`