elastisys:scale cloud pool REST API Documentation

Release 5.0.0

Elastisys AB

October 27, 2017

This documentation covers version 5.0.0 of the REST API that all elastisys:scale cloud pool endpoints are required to publish.

A *cloud pool* is a key component in an elastisys:scale autoscaling setup.

An elastisys:scale autoscaling system consists of two main parts: (1) an *autoscaler server* and (2) a *cloud pool*. The autoscaler server collects monitoring data reported by the application from a monitoring database and applies scaling algorithms to, ultimately, emit a number of required VM instances to keep the auto-scaling-enabled service running smoothly. This number is communicated over a secured (SSL) channel to the cloud pool, which instructs the cloud infrastructure to add or remove VMs, as appropriate. A schematical overview of the system is shown in the image below.

So a cloud pool manages an elastic pool of machines for a particular cloud provider, handling communication with the cloud provider according to its API. The cloud pool provides a cloud-neutral API to clients, such as the autoscaler, with a number of management primitives for the machine pool. In general terms, these primitives allow clients to:

- track the machine pool members and their states
- modify the size of the machine pool (the cloud pool continuously starts/stops machine instances so that the number of machines in the pool matches the desired size set for the pool).

The interface between the autoscaler and the cloud pool is a REST API. All cloud pool endpoints are required to implement this REST API and make it available over secure HTTP (HTTPS). The REST API is described in greater detail in the elastisys cloud pool REST API.

Documentation:

elastisys cloud pool REST API

All elastisys cloud pool endpoints are required to publish the REST API described below.

The primary purpose of the cloud pool API is to serve as a bridge between an autoscaler and a certain cloud provider, allowing the autoscaler to operate in a cloud-neutral manner. As such, it focuses on primitives for managing a dynamic collection of machines.

All API methods assume a `Content-Type` of `application/json`.

The REST API should be made available over secure HTTP (HTTPS).

Terminology and machine state model

Cloud providers differ in how they refer to the computational resources they provide. Some common terms are *instances*, *servers* and *VMs/virtual machines*.

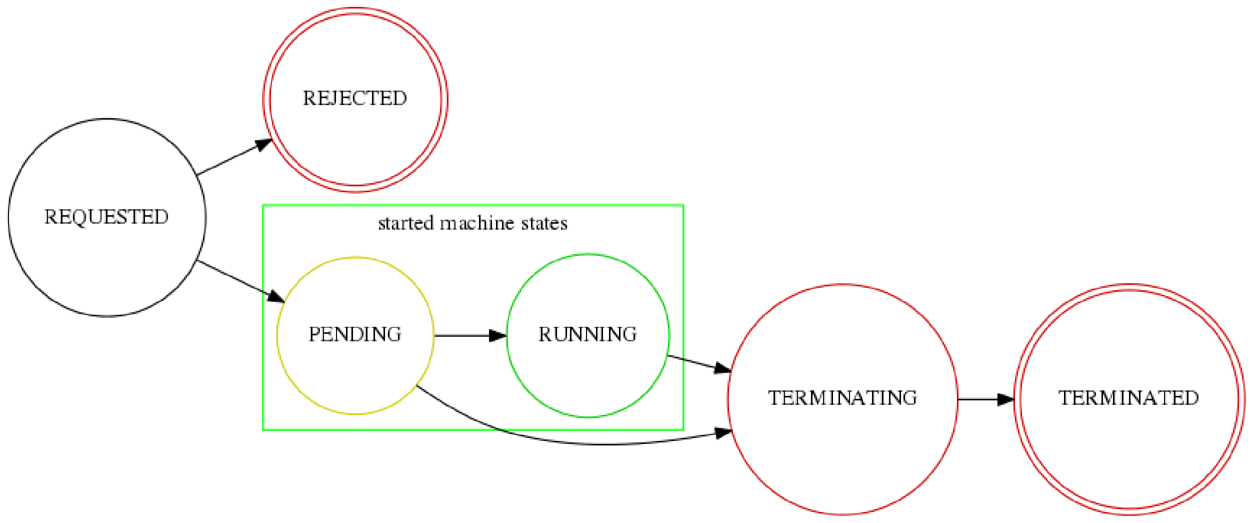
The cloud pool API strives to be as cloud-neutral as possible and simply refers to the computational resources being managed as **machines**. The logical group of machines that a cloud pool manages is referred to as its **machine pool**.

machine state

A cloud pool needs to be able to report the execution state of its machine pool members in a cloud-neutral manner (see *Get machine pool*). Since cloud providers differ quite a lot in the state models they use, the cloud pool needs to map the cloud-native state of the machine to one of the **machine states** supported by the cloud pool API. These states are described in the *machine state table* below.

Machine state	Description
`REQUESTED`	The machine has been requested from the underlying infrastructure and the request is pending fulfillment.
`REJECTED`	The machine request was rejected by the underlying infrastructure.
`PENDING`	The machine is in the process of being launched.
`RUNNING`	The machine is launched. However, the boot process may not yet have completed and the machine may not be operational (the machine's *service state* may provide more detailed state information).
`TERMINATING`	The machine is in the process of being stopped/shut down.
`TERMINATED`	The machine has been stopped/shut down.

The diagram below illustrates the state transitions that describe the lifecycle of a machine.



The `PENDING` and `RUNNING` states are said to be the *started machine states*. Machines in a started state are executing. However, just because a machine is executing doesn't necessarily mean that it is doing useful work. For example, it may have failed to properly boot, it may have crashed or encountered a fatal bug.

So the machine state is the execution state of the machine, as reported by the cloud API, which really only tells us if a particular pool member is started or not. To be able to reason about the *health* of a pool member, each machine's metadata carries two additional state fields – the *membership status* and the *service state*.

These states are intended to be set by external means, such as by a human operator or an external health monitoring service. A cloud pool is required to be ready to receive state updates these fields (see *Set membership status* and *Set service state*) for the machines its pool and to include those states on subsequent queries about the pool members (*Get machine pool*).

membership status

The **membership status** is used to indicate to the cloud pool that a certain machine needs to be given special treatment. The membership status can, for example, be set to protect a machine from being terminated (by setting its evictability) or to mark a machine as being in need of replacement (by setting its activity flag). This allows us, for example, to isolate a failed machine for further inspection and to provision a replacement to sustain sufficient capacity. It also allows us to have "blessed"/"seed" pool members that may not be terminated. See the *Set membership status* method for more deatils.

The `active` and `evictable` fields of the membership status can be combined according to the table below to produce four main membership states:

	active	**not active**
evictable	default	disposable
not evictable	blessed	awaiting service

- `default`: a machine that is both an active and evictable group member.
- `blessed`: a machine that is a permanent pool member that cannot be evicted. This can, for example, be used to include reserved machine instances in the pool.
- `awaiting service`: a machine that is in need of service. The machine is to be replaced and should be kept alive for troubleshooting.
- `disposable`: a machine that is non-functioning and should be replaced and terminated.

At any time, the *active size* of the cloud pool should be interpreted as the number of allocated machines that have not been marked with an inactive membership status. That is, all machines in one of the machine states `REQUESTED`, `PENDING`, or `RUNNING` and *not* having a membership status with `active` set to `false`.

service state

There are cases where we need to be able to reason about the operational state of the service running on the machine. For example, we may not want to register a running machine to a load balancer until it is fully initialized and ready to accept requests, and we may want to unregister unhealthy machines. To this end, a cloud pool may include a **service state** for a machine. Whereas the *machine state* should be viewed as the execution state of the *machine*, the *service state* should be viewed as the operational health of the *service running on the machine*. Service states have no semantic implications to the cloud pool. They should be regarded as informational "marker states" that may be used by third party services (such as a load balancer).

The range of permissible service states are as follows:

Service state	Description
`BOOTING`	The service is being bootstrapped and may not (yet) be
`IN_SERVICE`	The service is operational and ready to accept work (hea
`UNHEALTHY`	The service is not functioning properly (health checks fa
`OUT_OF_SERVICE`	The service is unhealthy and has been taken out of servi repair.
`UNKNOWN`	The service state of the machine cannot be (or has not y

See the *Set service state* method for more deatils.

Failure handling

Things fail in distributed systems. At different scales (network partitions, data center outage, cloud provider API errors, cloud pool misconfigurations, cloud pool bugs, etc) and on different time-scales (lasting for a single call, a few minutes to several hours). Since we live in a failure-prone world, cloud pool implementers are encouraged to temporarily mask cloud API failures to avoid prematurely panicking and propagating errors to upstream components.

In practice, a combination of measures can be taken to make a cloud pool less sensitive to cloud API errors:

- API calls can be retried a few times on failure (for example, with an exponential back-off algorithm). For example, cloud provider API calls could be retried a few times to guard against one-off API errors or short-lived error conditions.
- The cloud pool can be written to operate in an eventually consistent manner, periodically retrieving state data from the cloud API and storing it locally, responding to clients with the most recently observed state rather than synchronously trying to re-fetch the state from the cloud API on every invocation. The cloud pool should strive to provide "sufficiently up-to-date" data. At times when the cloud API is unreachable, it may suffice to serve somewhat dated data for a limited time, until fresh data can be retrieved again. When serving cached data, it is important to timestamp-mark the data so that the receiver of the data can determine if the data is fresh enough.

  Some operations are more suitable than others for this type of semantics (*Get machine pool*, *Get pool size*, *Set desired size*). Operations that are less suitable for this type of delayed semantics includes operations that act on individual machines (*Terminate machine*, *Attach machine*, *Detach machine*, *Set service state*, *Set membership status*). For such operations, it makes more sense to immediately try to "write through" to the cloud API and respond with an error on failure.

Operations

Set configuration

- **Method**: `POST /config`
- **Description**: Sets a new configuration for the cloud pool.

  The configuration is a JSON document whose appearance depends on the particular cloud pool implementation.

  This operation will not change the cloud pool's started state – if the cloud pool had been started (see *Start*) it will remain started, and if it was in a stopped state it will remain stopped.
- **Input**: A JSON document with a configuration that follows the schema of the particular cloud pool implementation.
- **Output:**
  - on success: HTTP response code 200.
  - on invalid input (for example, if the cloud pool fails to validate the configuration): HTTP response code 400 (Bad Request) with an *Error response message*.
  - cloud API errors: HTTP response code 502 (Bad Gateway) with an *Error response message*
  - other errors: HTTP response code 500 (Internal Server Error) with an *Error response message*

Get configuration

- **Method**: `GET /config`
- **Description**: Retrieves the configuration currently set for the cloud pool (if any).

  The configuration is a JSON document whose appearance depends on the particular cloud pool implementation.
- **Input**: None
- **Output:**
  - HTTP response code 200 with a configuration JSON document on success.
  - HTTP response code 404 (Not Found) if no configuration has been set.
  - On error: HTTP response code 500 (Internal Server Error) with an *Error response message*

Start

- **Method**: `POST /start`
- **Description**: Starts the cloud pool.

  This will set the cloud pool in an activated state where it will start to accept requests to query or modify the machine pool.

  If the cloud pool has not been configured (see *Set configuration*) the method will fail. If the cloud pool is already started this is a no-op.
- **Input**: None
- **Output:**
  - HTTP response code 200 on success.

- HTTP response code 400 (Bad Request) with an *Error response message* on an attempt to start an unconfigured cloud pool.
- HTTP response code 500 (Internal Server Error) with an *Error response message* on error.

Stop

- **Method**: `POST /stop`
- **Description**: Stops the cloud pool.

  A stopped cloud pool is in a passivated state and will not accept any requests to query or modify the machine pool.

  If the cloud pool is already in a stopped state this is a no-op.
- **Input**: None
- **Output:**
  - HTTP response code 200 on success.
  - HTTP response code 500 (Internal Server Error) with an *Error response message* on error.

Get status

- **Method**: `GET /status`
- **Description**: Retrieves the execution status for the cloud pool.
- **Input**: None
- **Output:**
  - HTTP response code 200 on success with a *Status message*.
  - HTTP response code 500 (Internal Server Error) with an *Error response message* on error.

Get machine pool

- **Method**: `GET /pool`
- **Description**: Retrieves the current machine pool members.

  Note that the returned machines may be in any *machine state* (`REQUESTED`, `RUNNING`, `TERMINATED`, etc).

  The *membership status* of a started machine determines if it is to be considered an active member of the pool.The *active size* of the machine pool should be interpreted as the number of allocated machines (in any of the non-terminal machine states `REQUESTED`, `PENDING` or `RUNNING` that have not been marked with an inactive *membership status*.

  The *service state* should be set to `UNKNOWN` for all machine instances for which no service state has been reported (see *Set service state*).

  Similarly, the *membership status* should be set to the default (active, evictable) status for all machine instances for which no membership status has been reported (see *Set membership status*).
- **Input**: None
- **Output:**
  - On success: HTTP response code 200 with a *Machine pool message*

  - On cloud API errors: HTTP response code 502 (Bad Gateway) with an *Error response message*
  - On other errors: HTTP response code 500 (Internal Server Error) with an *Error response message*

Get pool size

- **Method**: `GET /pool/size`
- **Description**: Returns the current size of the machine pool – both in terms of the desired size and the actual size (as these may differ at any time).
- **Input**: None
- **Output:**
  - On success: HTTP response code 200 with a *Pool size message*
  - On cloud API errors: HTTP response code 502 (Bad Gateway) with an *Error response message*
  - On other errors: HTTP response code 500 (Internal Server Error) with an *Error response message*

Set desired size

- **Method**: `POST /pool/size`
- **Description**: Sets the desired number of machines in the machine pool. This method is asynchronous and returns immediately after updating the desired size. There may be a delay before the changes take effect and are reflected in the machine pool.

  Note: the cloud pool should take measures to ensure that requested machines are recognized as pool members. The specific mechanism to mark group members, which may depend on the features offered by the particular cloud API, is left to the implementation but could, for example, make use of tags.
- **Input**: The desired number of machine instances in the pool as a *Set desired size message*.
- **Output:**
  - On success: HTTP response code 200 without message content.
  - **On error:**
    - on illegal input: code 400 with an *Error response message*
    - otherwise: HTTP response code 500 (Internal Server Error) with an *Error response message*

Terminate machine

- **Method**: `POST /pool/terminate`
- **Description**: Terminates a particular machine pool member. The caller can control if a replacement machine is to be provisioned via the `decrementDesiredSize` parameter.

  Note: a machine that is protected from removal by a membership status with `evictable: false` can not be terminated.
- **Input**: A *Terminate machine message*.
- **Output:**
  - On success: HTTP response code 200 without message content.
  - **On error:**

    * on illegal input: code 400 with an *Error response message*
    * if the machine is not a pool member: code 404 with an *Error response message*
    * on cloud API errors: HTTP response code 502 (Bad Gateway) with an *Error response message*
    * On other errors: HTTP response code 500 (Internal Server Error) with an *Error response message*

Set membership status

- **Method**: `POST /pool/membershipStatus`
- **Description**: Sets the *membership status* of a given pool member.

  The membership status for a machine can be set to protect the machine from being terminated (by setting its evictability status) and/or to mark a machine as being in need of replacement by flagging it as an inactive pool member.

  The specific mechanism to mark group members, which may depend on the features offered by the particular cloud API, is left to the implementation but could, for example, make use of tags.
- **Input**: A *Set membership status message*.
- **Output:**
  - On success: HTTP response code 200 without message content.
  - **On error:**
    * on illegal input: code 400 with an *Error response message*
    * if the machine is not a pool member: code 404 with an *Error response message*
    * on cloud API errors: HTTP response code 502 (Bad Gateway) with an *Error response message*
    * On other errors: HTTP response code 500 (Internal Server Error) with an *Error response message*

Set service state

- **Method**: `POST /pool/serviceState`
- **Description**: Sets the *service state* of a given machine pool member.

  Setting the service state does not have any functional implications on the pool member, but should be seen as way to supply operational information about the service running on the machine to third-party services (such as load balancers).

  The specific mechanism to mark group members, which may depend on the features offered by the particular cloud API, is left to the implementation but could, for example, make use of tags.
- **Input**: A *Set service state message*.
- **Output:**
  - On success: HTTP response code 200 without message content.
  - **On error:**
    * on illegal input: code 400 with an *Error response message*
    * if the machine is not a pool member: code 404 with an *Error response message*

    * on cloud API errors: HTTP response code 502 (Bad Gateway) with an *Error response message*
    * On other errors: HTTP response code 500 (Internal Server Error) with an *Error response message*

Detach machine

- **Method**: `POST /pool/detach`
- **Description**: Removes a particular machine pool member from the pool without terminating it.

  The machine keeps running but is no longer considered a pool member and, therefore, needs to be managed independently. The caller can control if a replacement machine is to be provisioned via the `decrementDesiredSize` parameter.

  Note: a machine that is protected from removal by a membership status with `evictable: false` can not be detached.
- **Input**: A *Detach machine message*.
- **Output:**
  - On success: HTTP response code 200 without message content.
  - **On error:**
    * on illegal input: code 400 with an *Error response message*
    * if the machine is not a pool member: code 404 with an *Error response message*
    * on cloud API errors: HTTP response code 502 (Bad Gateway) with an *Error response message*
    * On other errors: HTTP response code 500 (Internal Server Error) with an *Error response message*

Attach machine

- **Method**: `POST /pool/attach`
- **Description**: Attaches an already running machine to the machine pool, growing the pool with a new member. This operation implies that the desired size of the group is incremented by one.
- **Input**: An *Attach machine message*.
- **Output:**
  - On success: HTTP response code 200 without message content.
  - **On error:**
    * on illegal input: code 400 with an *Error response message*
    * if the machine does not exist: code 404 with an *Error response message*
    * on cloud API errors: HTTP response code 502 (Bad Gateway) with an *Error response message*
    * On other errors: HTTP response code 500 (Internal Server Error) with an *Error response message*

Messages

Status message

Description	A message used to report the state of the cloud pool.

The status message has the following schema:

```
{
  "started": <boolean>,
  "configured": <boolean>
}
```

Sample document:

```
{
  "started": true,
  "configured": true
}
```

Set desired size message

Description	A message used to request that the machine pool be resized to a desired number of machine instances.
Schema	`{ "desiredSize":  <number> }`

Sample document:

```
{ "desiredSize": 3 }
```

States that we want three machine instances in the pool.

Error response message

Description	Contains further details (in addition to the HTTP response code) on server-side errors.
Schema	`{ "message":  <string>, "detail":  <string> }`

The `message` is a human-readable error message intended for presentation, whereas the `detail` attribute holds error details (such as a stack trace).

This is a sample error message:

```
{
  "message": "failure to process pool get request",
  "detail": "... long stacktrace ..."
}
```

Machine pool message

Description	Describes the current status of the monitored machine pool.

The machine pool schema has the following structure:

```
{
  "timestamp": <iso-8601 datetime>,
  "machines": [ <machine> ... ]
}
```

The `timestamp` is the time at which the pool observation was made. Note that in case the cloud pool serves locally cached data, this field may be used by the client to determine if the data is fresh enough to be acted upon.

Every `<machine>` entry is a json object with the following structure:

```
{
  "id":               <string>,
  "machineState":     <machine state>,
  "membershipStatus": {"active": bool, "evictable": bool},
  "serviceState":     <service state>,
  "cloudProvider":    <string>,
  "region":           <string>,
  "machineSize":      <string>,
  "launchTime":       <iso-8601 datetime>,
  "requestTime":      <iso-8601 datetime>,
  "publicIps":        [<ip-address>, ...],
  "privateIps":       [<ip-address>, ...],
  "metadata":         <jsonobject>
}
```

The attributes are to be interpreted as follows:

- `id`: The identifier of the machine.
- `machineState`: The execution state of the machine. See the section on *machine state*.
- `membershipStatus`: The *membership status* of the machine.
- `serviceState`: The operational state of the service running on the machine. See the section on *service state*.
- `cloudProvider`: The name of the cloud provider that this machine originates from, for example *AWS-EC2*. It might not be immediately apparent why this field is required since the cloud pool itself states which cloud provider it supports, but it is useful to distinguish where different machines originate from in multi-cloud scenarios where multiple down-stream cloud pools are abstracted by an upstream aggregating cloud pool (such as a splitter pool).
- `region`: The name of the cloud region/zone/data center where this machine is located. For example, *us-east-1*. As for the `cloudProvider` attribute, this attribute can be useful to an upstream component that collects machines from multiple cloud pools.
- `machineSize`: The size of the machine (or instance type, in Amazon EC2 terminology). For example, *m1.medium* for an Amazon EC2 machine.
- `requestTime`: The request time of the machine if one can be determined by the underlying infrastructure. Since not all infrastructures support this, it may be left out or set to `null`.
- `launchTime`: The launch time of the machine if it has been launched. If the machine is in a state where it hasn't been launched yet (`REQUESTED` state) this attribute may be left out or set to `null`.
- `publicIps`: The list of public IP addresses associated with this machine. Depending on the state of the machine, this list may be empty.
- `privateIps`: The list of private IP addresses associated with this machine. Depending on the state of the machine, this list may be empty.
- `metadata`: Additional cloud provider-specific meta data about the machine. This field is optional (may be `null`).

Below is a sample machine pool document:

```
{
  "timestamp": "2013-11-07T13:50:00.000Z",
  "machines": [
    {
      "id": "i-123456",
      "machineState": "RUNNING",
      "cloudProvider": "AWS_EC2",
      "region": "us-east-1",
      "machineSize": "m1.small",
      "membershipStatus": {"active": true, "evictable": true},
      "serviceState": "IN_SERVICE",
      "requestTime": "2013-11-07T14:48:00.000Z",
      "launchTime": "2013-11-07T14:50:00.000Z",
      "publicIps": ["54.211.230.169"],
      "privateIps": ["10.122.122.69"],
      "metadata": {
        "scaling-group": "mygroup"
      }
    },
    {
      "id": "i-123457",
      "machineState": "PENDING",
      "cloudProvider": "AWS_EC2",
      "region": "us-east-1",
      "machineSize": "m1.small",
      "membershipStatus": {"active": true, "evictable": true},
      "serviceState": "BOOTING",
      "requestTime": "2013-11-07T13:47:50.000Z",
      "launchTime": "2013-11-07T13:49:50.000Z",
      "publicIps": [],
      "privateIps": [],
      "metadata": {
        "scaling-group": "mygroup",
      }
    }
  ]
}
```

Pool size message

Description	Carries information about the pool size, both desired and actual size.
Schema	`{ "timestamp": "<time>", "desiredSize": <number>, "allocated": <number>, "active": <number> }`

The attributes are to be interpreted as follows:

- `timestamp`: The time at which the pool size observation was made. Note that in case the cloud pool serves locally cached data, this field may be used by the client to determine if the data is fresh enough to be acted upon.
- `desiredSize`: The last desired size set for the machine pool (see *Set desired size*).
- `allocated`: The number of allocated machines in the pool (in one of machine states `REQUESTED`, `PENDING`, `RUNNING`)
- `active`: The number of machines in the pool with an `active` *membership status*.

Example:

```
{ "timestamp": "2015-01-01T12:50:00.000Z", "desiredSize": 3, "allocated": 4, "active": 3 }
```

Terminate machine message

Description	Specifies which pool member to terminate and if the desired size of the machine pool should be decremented after terminating the machine (that is, it controls if a replacement machine should be launched)
Schema	`{ "machineId": "<id>", decrementDesiredSize": <boolean> }`

The attributes are to be interpreted as follows:

- `machineId`: The (cloudprovider-specific) id of the machine to terminate.
- `decrementDesiredSize`: `true` if the desired pool size should be decremented, `false` otherwise.

Example where a replacement machine is desired:

```
{ "machineId": "i-123457", "decrementDesiredSize": false }
```

Detach machine message

Description	Specifies which pool member to detach and if the desired size of the machine pool should be decremented after detaching the machine (that is, it controls if a replacement machine should be launched)
Schema	`{ "machineId": "<id>", "decrementDesiredSize": <boolean> }`

The attributes are to be interpreted as follows:

- `machineId`: The (cloudprovider-specific) id of the machine to detach.
- `decrementDesiredSize`: `true` if the desired pool size should be decremented, `false` otherwise.

Example where a replacement machine is desired:

```
{ "machineId": "i-123457", "decrementDesiredSize": false }
```

Attach machine message

Description	Specifies a cloud machine that is to be attached to the cloudpool.
Schema	`{ "machineId": "<id>" }`

The attributes are to be interpreted as follows:

- `machineId`: The (cloudprovider-specific) id of the machine to attach.

Example:

```
{ "machineId": "i-123457" }
```

Set membership status message

Description	Updates the membership status for a machine.
Schema	`{ "machineId": "<id>", "membershipStatus": {"active": bool, "evictable": bool} }`

The attributes are to be interpreted as follows:

- `machineId`: The (cloudprovider-specific) id of the machine to update.
- `active`: Indicates if this is an active (working) pool member. A `true` value indicates that this machine is a functioning pool member. A `false` value indicates that a replacement machine needs to be launched for this pool member.
- `evictable`: Indicates if this machine is a blessed member of the machine pool. That is, if this field is `true`, the cloud pool may not select this machine for termination when pool needs to be scaled in.

Example of a membership status for a broken machine that needs a replacement (`active` == `false`), but is to be kept around in the pool for troubleshooting (`evictable` == `false`):

```
{ "machineId": "i-123457", "membershipStatus": {"active": false, "evictable": false} }
```

Set service state message

Description	Updates the service state for a particular pool member.
Schema	`{ "machineId": "<id>", "serviceState": "<service state>" }`

The attributes are to be interpreted as follows:

- `machineId`: The (cloudprovider-specific) id of the machine to update.
- `serviceState`: The *service state* to set.

Example where a replacement machine is desired:

```
{ "machineId": "i-123457", "serviceState": "IN_SERVICE" }
```