# Nodepool

## **Install Nodepool**

Install Nodepool prerequisites.

Nodepool requires Python 3.6 or newer.

**RHEL 7 / CentOS 7:**

**yum install libffi libffi-devel @development python python-devel**

You may install Nodepool directly from PyPI with pip:

**pip install nodepool**

Or install directly from a git checkout with:

**pip install .**

## **Configuration**

Nodepool has one required configuration file, which defaults to /etc/nodepool/nodepool.yaml. This can be changed with the -c option.

There is support for a secure file that is used to store nodepool configurations that contain sensitive data. It currently only supports specifying ZooKeeper credentials and diskimage env-vars. If ZooKeeper credentials or diskimage env-vars are defined in both configuration files, the data in the secure file takes precedence. The secure file location can be changed with the -s option and follows the same file format as the Nodepool configuration file.

## **Nodepool-builder**

The nodepool-builder daemon builds and uploads images to providers. It may be run on the same or a separate host as the main nodepool daemon. Multiple instances of nodepool-builder may be run on the same or separate hosts in order to speed up image builds across many machines, or supply high-availability or redundancy. However, since nodepool-builder allows specification of the number of both build and upload threads, it is usually not advantageous to run more than a single instance on one machine. Note that while diskimage-builder (which is responsible for building the underlying images) generally supports executing multiple builds on a single machine simultaneously, some of the elements it uses may not. To be safe, it is recommended to run a single instance of nodepool-builder on a machine, and configure that instance to run only a single build thread (the default).

## **Nodepool-launcher**

The main nodepool daemon is named nodepool-launcher and is responsible for managing cloud instances launched from the images created and uploaded by nodepool-builder.

When a new image is created and uploaded, nodepool-launcher will immediately start using it when launching nodes (Nodepool always uses the most recent image for a given provider in the ready state). Nodepool will delete images if they are not the most recent or second most recent ready images. In other words, Nodepool will always make sure that in addition to the current image, it keeps the previous image around. This way if you find that a newly created image is problematic, you may simply delete it and Nodepool will revert to using the previous image.

## **Common Management Tasks**

In the course of running a Nodepool service you will find that there are some common operations that will be performed. Like the services themselves these are split into two groups, image management and instance management.

### **Image Management**

Before Nodepool can launch any cloud instances it must have images to boot off of. nodepool dib-image-list will show you which images are available locally on disk. These images on disk are then uploaded to clouds, nodepool image-list will show you what images are bootable in your various clouds.

If you need to force a new image to be built to pick up a new feature more quickly than the normal rebuild cycle (which defaults to 24 hours) you can manually trigger a rebuild. Using nodepool image-build you can tell Nodepool to begin a new image build now. Note that depending on work that the nodepool-builder is already performing this may queue the build. Check nodepool dib-image-list to see the current state of the builds. Once the image is built it is automatically uploaded to all of the clouds configured to use that image.

At times you may need to stop using an existing image because it is broken. Your two major options here are to build a new image to replace the existing image or to delete the existing image and have Nodepool fall back on using the previous image. Rebuilding and uploading can be slow so typically the best option is to simply nodepool image-delete the most recent image which will cause Nodepool to fallback on using the previous image. Howevever, if you do this without “pausing” the image it will be immediately reuploaded. You will want to pause the image if you need to further investigate why the image is not being built correctly. If you know the image will be built correctly you can simple delete the built image and remove it from all clouds which will cause it to be rebuilt using nodepool dib-image-delete.

Nodepool has two components which run as daemons. The nodepool-builder daemon is responsible for building diskimages and uploading them to providers, and the nodepool-launcher daemon is responsible for launching and deleting nodes.

Both daemons frequently re-read their configuration file after starting to support adding or removing new images and providers, or otherwise altering the configuration.

These daemons communicate with each other via a Zookeeper database. You must run Zookeeper and at least one of each of these daemons to have a functioning Nodepool installation.

## **Web interface**

If configured (see webapp), a nodepool-launcher instance can provide a range of end-points that can provide information in text and json format. Note if there are multiple launchers, all will provide the same information.

**GET /image-list**

The status of uploaded images

**Query Parameters:**

* **fields** – comma-separated list of fields to display

**Request Headers:**

* Accept – application/json or text/\*

**Response Headers:**

* Content-Type – application/json or text/plain depending on the Accept header

**GET /dib-image-list**

The status of images built by diskimage-builder

**Query Parameters:**

* **fields** – comma-separated list of fields to display

**Request Headers:**

* Accept – application/json or text/\*

**Response Headers:**

* Content-Type – application/json or text/plain depending on the Accept header

**GET /image-status**

The paused and manual build status of images

**Query Parameters:**

* **fields** – comma-separated list of fields to display

**Request Headers:**

* Accept – application/json or text/\*

**Response Headers:**

* Content-Type – application/json or text/plain depending on the Accept header

**GET /node-list**

The status of currently active nodes

**Query Parameters:**

* **node\_id** – restrict to a specific node
* **fields** – comma-separated list of fields to display

**Request Headers:**

* Accept – application/json or text/\*

**Response Headers:**

* Content-Type – application/json or text/plain depending on the Accept header

**GET /request-list**

Outstanding requests

**Query Parameters:**

* **fields** – comma-separated list of fields to display

**Request Headers:**

* Accept – application/json or text/\*

**Response Headers:**

* Content-Type – application/json or text/plain depending on the Accept header

**GET /label-list**

All available labels as reported by all launchers

**Query Parameters:**

* **fields** – comma-separated list of fields to display

**Request Headers:**

* Accept – application/json or text/\*

**Response Headers:**

* Content-Type – application/json or text/plain depending on the Accept header

**GET /ready**

Responds with status code 200 as soon as all configured providers are fully started. During startup it returns 500. This can be used as a readiness probe in a kubernetes based deployment.

## **Monitoring**

Nodepool provides monitoring information to statsd.

### **Nodepool builder**

The following metrics are produced by a nodepool-builder process:

**nodepool.dib\_image\_build.<diskimage\_name>.<ext>.size*(gauge)***

This stat reports the size of the built image in bytes. ext is based on the formats of the images created for the build, for example qcow2, raw, vhd, etc.

**nodepool.dib\_image\_build.<diskimage\_name>.status.rc*(gauge)***

Return code of the last DIB run. Zero is successful, non-zero is unsuccessful.

**nodepool.dib\_image\_build.<diskimage\_name>.status.duration*(timer)***

Time the last DIB run for this image build took, in ms

**nodepool.dib\_image\_build.<diskimage\_name>.status.last\_build*(gauge)***

The UNIX timestamp of the last time a build for this image returned. This can be useful for presenting a relative time (“X hours ago”) in a dashboard.

**nodepool.image\_update.<image name>.<provider name>*(counter,* *timer)***

Number of image uploads to a specific provider in the cloud plus the time in ms spent to upload the image.

**nodepool.image\_build\_requests*(gauge)***

Number of manual build requests outstanding (does not include currently running builds).

### **Nodepool launcher**

The following metrics are produced by a nodepool-launcher process:

**nodepool.nodes.<state>*(counter)***

Number of nodes in a specific state.

state can be:

* building
* deleting
* failed
* in-use
* ready
* used

**nodepool.label.<label>.nodes.<state>*(counter)***

Number of nodes with a specific label in a specific state. See nodepool.nodes for a list of possible states.

**nodepool.tenant\_limits.<tenant>.<limit>*(guage)***

The currently configured resource limits of a tenant.

limit can be:

* cores
* instances
* ram

### **Provider Metrics**

**nodepool.provider.<provider>.max\_servers*(gauge)***

Current setting of the max-server configuration parameter for the respective provider.

**nodepool.provider.<provider>.nodes.<state>*(gauge)***

Number of nodes per provider that are in one specific state. See nodepool.nodes for a list of possible states.

**nodepool.provider.<provider>.leaked**

This hierarchy supplies driver-dependent information about leaked resource cleanup. Non-zero values indicate an error situation as resources should be cleaned up automatically.

**nodepool.provider.<provider>.leaked.amis*(counter)***

Drivers: AWS

Number of leaked AMIs removed automatically by Nodepool.

**nodepool.provider.<provider>.leaked.disks*(counter)***

Drivers: Azure

Number of leaked disks removed automatically by Nodepool.

**nodepool.provider.<provider>.leaked.floatingips*(counter)***

Drivers: OpenStack, IBMVPC

Number of unattached floating IPs removed automatically by Nodepool.

**nodepool.provider.<provider>.leaked.images*(counter)***

Drivers: Azure, IBMVPC

Number of leaked images removed automatically by Nodepool.

**nodepool.provider.<provider>.leaked.instances*(counter)***

Drivers: AWS, Azure, GCE, IBMVPC, OpenStack

Number of nodes not correctly recorded in Zookeeper that Nodepool has cleaned up automatically.

**nodepool.provider.<provider>.leaked.nics*(counter)***

Drivers: Azure

Number of leaked NICs removed automatically by Nodepool.

**nodepool.provider.<provider>.leaked.objects*(counter)***

Drivers: AWS, IBMVPC

Number of leaked storage objects removed automatically by Nodepool.

**nodepool.provider.<provider>.leaked.pips*(counter)***

Drivers: Azure

Number of leaked public IPs removed automatically by Nodepool.

**nodepool.provider.<provider>.leaked.ports*(counter)***

Drivers: OpenStack

Number of ports in the DOWN state that have been removed.

**nodepool.provider.<provider>.leaked.snapshots*(counter)***

Drivers: AWS

Number of leaked snapshots removed automatically by Nodepool.

**nodepool.provider.<provider>.leaked.volumes*(counter)***

Drivers: AWS

Number of leaked volumes removed automatically by Nodepool.

### **Launch metrics**

**nodepool.launch.<result>*(counter,* *timer)***

Number of launches, categorized by the launch result plus the duration of the launch.

result can be:

* ready: launch was successful
* error.zksession: Zookeeper session was lost
* error.quota: Quota of the provider was reached
* error.unknown: Some other error during launch

**nodepool.launch.provider.<provider>.<az>.<result>*(counter,* *timer)***

Number of launches per provider per availability zone, categorized by the launch result plus duration of the launch.

See nodepool.launch for a list of possible results.

**nodepool.launch.image.<image>.<result>*(counter,* *timer)***

Number of launches per image, categorized by the launch result plus duration of the launch.

See nodepool.launch for a list of possible results.

**nodepool.launch.requestor.<requestor>.<result>*(counter,* *timer)***

Number of launches per requestor, categorized by the launch result plus the duration of the launch.

See nodepool.launch for a list of possible results.