

---

# **Solum Documentation**

*Release 10.0.0.0rc2.dev2*

**OpenStack Foundation**

**Apr 06, 2021**



# CONTENTS

<b>1</b>	<b>Solum Quick Start Guide</b>	<b>3</b>
1.1	Setup Solum development environment	3
1.2	Overview	3
1.3	Create a languagepack	3
1.4	Create your app	5
1.5	Deploy your app	6
1.6	Connect to Your App	8
1.7	Update Your App	8
1.8	Languagepacks	9
1.9	appfile	9
1.10	App configuration and environment variables	10
1.11	Set up a Development Environment	10
1.12	Vagrant Dev Environment	10
1.13	Devstack	11
<b>2</b>	<b>Install Solum</b>	<b>13</b>
2.1	Distro specific installation	13
2.2	For a development installation use devstack	13
<b>3</b>	<b>Enabling Solum in DevStack</b>	<b>15</b>
<b>4</b>	<b>Configure and run Solum</b>	<b>17</b>
4.1	Configuration Reference	17
4.2	Administrator Guide	17
4.2.1	Man pages for services and utilities	17
	Solum utilities	17
4.3	High Availability Guide	19
4.4	Operations Guide	19
4.5	Security Guide	20
<b>5</b>	<b>Develop applications for Solum</b>	<b>21</b>
5.1	API Complete Reference	21
5.1.1	Version discovery	21
5.1.2	V1 API	22
	Platform	22
	Plans	24
	Pipelines	26
	Executions	28
	Assemblies	28

Services . . . . .	30
Operations . . . . .	31
Sensors . . . . .	33
Components . . . . .	34
Extensions . . . . .	36
LanguagePacks . . . . .	38
Infrastructure . . . . .	41
Triggers . . . . .	42
<b>6 How to contribute to Solum</b>	<b>43</b>
<b>7 CLI Reference</b>	<b>45</b>
7.1 solum-status . . . . .	45
7.1.1 Synopsis . . . . .	45
7.1.2 Description . . . . .	45
7.1.3 Options . . . . .	45
Upgrade . . . . .	46

Contents:



## SOLUM QUICK START GUIDE

### 1.1 Setup Solum development environment

<https://wiki.openstack.org/wiki/Solum/solum-development-setup>

The following is a guide to deploying an app with Solum.

### 1.2 Overview

```
$ solum languagepack create <NAME> <GIT_REPO>
$ solum languagepack show <UUID/Name>
$ solum languagepack logs <UUID>
$ solum languagepack list
$ solum app create --app-file <app_file> [--param-file param_file]
$ solum app show <UUID/Name>
$ curl <application_uri>
```

In this document we will work with a python example to demonstrate how you can use solum to deploy an application.

### 1.3 Create a languagepack

Before deploying an app on Solum, we need to create a run time environment, called languagepack, for the application. A languagepack must exist in Solum, as every application deployed with Solum requires an association to a languagepack to run (even if the languagepack only implements a no-op). Languagepacks can be added to Solum in the following ways:

1. Solum comes with pre-existing languagepacks
2. Solum System Operator creates and adds languagepack(s) available for all users
3. Solum User creates and adds languagepack(s) available only to that user

To learn more, see the languagepacks section of this document.

1. Authenticate to Keystone. The easiest way is to use the credentials supplied by Devstack.

```
$ . ~/devstack/openrc
```

2. Create languagepack

```
$ solum languagepack create python https://github.com/rackspace-solum-
↳samples/solum-languagepack-python.git

+-----+
↳-----+
| Property  | Value
↳
+-----+
↳-----+
| status    | QUEUED
↳
| source_uri | https://github.com/rackspace-solum-samples/solum-
↳languagepack-python.git |
| description | None
↳
| uuid      | 0233f461-5fb0-4de7-8f06-5527721c3e97
↳
| name      | python
↳
+-----+
↳-----+
```

Solum takes a few minutes to build your languagepack. You can check the state by using the languagepack show command. A languagepack is ready for use once the state changes to 'READY'.

```
$ solum languagepack show python

+-----+
↳-----+
| Property  | Value
↳
+-----+
↳-----+
| status    | READY
↳
| source_uri | https://github.com/rackspace-solum-samples/solum-
↳languagepack-python.git |
| description | None
↳
| uuid      | 0233f461-5fb0-4de7-8f06-5527721c3e97
↳
| name      | python
↳
+-----+
↳-----+
```

You can check logs that were generated while building the languagepack with the following command. This is a great way to debug your languagepack if it fails to build.

```
$ solum languagepack logs python

+-----+-----+-----+
↳-----+
| resource_uuid          | created_at          | local_
↳storage
+-----+-----+-----+
↳-----+
| 0233f461-5fb0-4de7-8f06-5527721c3e97 | 2016-04-07 13:33:35 | /var/log/
↳solum/worker/languagepack=2a8cd98e-8b37-4ec7-b17b-f511814a7d6f_log |
(continues on next page)
```



(continued from previous page)

```
+-----+-----+-----+
↪-----+
```

You can find all available languagepacks with the following command

```
$ solum languagepack list
+-----+-----+-----+-----+
↪-----+
| uuid          | name      | description | status |
↪source_uri    |           |             |        |
↪|             |           |             |        |
+-----+-----+-----+-----+
↪-----+
| 95310b74-b3ed-4150-b0bf-e64c21359900 | java      | None        | READY |
↪https://github.com/rackspace-solum-samples/solum-languagepack-java.git
↪|
| 96f889e7-e8db-4ae3-a38d-0bfd8268e30 | python    | None        | READY |
↪https://github.com/rackspace-solum-samples/solum-languagepack-python.git
↪|
+-----+-----+-----+-----+
↪-----+
```

## 1.4 Create your app

Solum clones code from the user's public Git repository or user's public/private GitHub repository. Before you begin, push your code to a Git repo. From within your devstack host, you can now run solum commands to build and deploy your application.

2. To register an app with Solum, you will need to write an appfile to describe it. The following appfile deploys a sample python application. You can find other examples in the `examples/apps/` folder of the solum repo on github. To learn more, see the appfile section of this document.

```
version: 1
name: cherrypy
description: python web app
languagepack: python
source:
  repository: https://github.com/rackspace-solum-samples/solum-python-
↪sample-app.git
  revision: master
workflow_config:
  test_cmd: ./unit_tests.sh
  run_cmd: python app.py
trigger_actions:
- unittest
- build
- deploy
ports:
- 80
```

The app is named `cherrypy`, and it describes a single application, running the code from the given Github repo. The code in that repo is a Python app that listens for HTTP requests and returns environment variables supplied by the user during app creation. We have configured this example to listen on

port 80.

## 1.5 Deploy your app

3. Create an app by supplying the appfile. This registers your app with Solum. For demonstration purposes, we will use the provided example.

```
$ solum app create --app-file appfile.yaml --param-file params.yaml
+-----+-----+
| Property | Value |
+-----+-----+
| description | Sample Python web app. |
+-----+-----+
| uri | http://10.0.2.15:9777/v1/plans/4a795b99-936d-4330-be4d-d2099b160075 |
+-----+-----+
| name | cherryppy |
+-----+-----+
| trigger_uri | |
+-----+-----+
| uuid | 4a795b99-936d-4330-be4d-d2099b160075 |
+-----+-----+
```

The `uri` field above refers to the newly-registered app. At this point, your app is not deployed yet.

Your app is now ready to be deployed using the `uuid` from above to deploy your app.

### 4. Deploy app

```
$ solum app deploy 4a795b99-936d-4330-be4d-d2099b160075
+-----+-----+
| Property | Value |
+-----+-----+
| wf_id | 1 |
+-----+-----+
| created_at | 2016-04-07T13:36:45.497519 |
+-----+-----+
| app_id | 7d64347c-93d6-4adf-bf70-309f9d53c034 |
+-----+-----+
| actions | [u'unittest', u'build', u'deploy'] |
+-----+-----+
| updated_at | 2016-04-07T13:36:45.497519 |
+-----+-----+
| source | {u'repository': u'https://github.com/rackspace-solum- |
| samples/solum- | python-sample-app.git', u'revision': u'master'} |
+-----+-----+
| config | {u'run_cmd': u'python app.py', u'test_cmd': u'./unit_tests. |
| sh'} |
+-----+-----+
```

(continues on next page)

(continued from previous page)

id	97e7e2c1-8ba1-4320-9831-b5baef1d480d	
↪		
+-----+	+-----+	+-----+
↪-----+		

Solum builds a Docker image by layering your app's code on top of the related languagepack's docker image. Then, Solum creates a stack via Heat to deploy your app. At this point, Solum is done, and in a matter of minutes your app will be deployed.

5. You can monitor the progress of your app as it builds and deploys. The status field will show the progress of your app through the process.

```
$ solum app show 4a795b99-936d-4330-be4d-d2099b160075
```

+-----+	+-----+	+-----+
↪-----+		
Property	Value	
↪		
+-----+	+-----+	+-----+
↪-----+		
status	BUILDING	
↪		
description	Sample Python web app.	
↪		
application_uri	None	
↪		
created_at	2015-03-10T22:47:04	
↪		
updated_at	2015-03-10T22:49:59	
↪		
name	cherryypy	
↪		
trigger_uri	http://10.0.2.15:9777/v1/triggers/b6eb26e5-3b7b-416b-	
↪b932-302c514071cc		
uuid	185f2741-61e0-497e-b2b7-c890c7e151dd	
↪		
+-----+	+-----+	+-----+
↪-----+		

6. Run the `solum app show` command a few times to see the status change. You will notice the status field changes to `DEPLOYMENT_COMPLETE` and the `application_uri` is available.

```
$ solum app show cherryypy
```

+-----+	+-----+	+-----+
↪-----+		
Property	Value	
↪		
+-----+	+-----+	+-----+
↪-----+		
app_url	172.24.4.3:80	
↪		
entry_points		
↪		
description	python web app	
↪		

(continues on next page)

(continued from previous page)

```

| created_at          | 2016-04-07T13:36:32
↪
| languagepack       | python
↪
| target_instances   | 1
↪
| ports              | [80]
↪
| source
↪samples/solum-     | {u'repository': u'https://github.com/rackspace-solum-
|                    | python-sample-app.git', u'revision': u'master'}
↪
| trigger            | [u'unittest', u'build', u'deploy']
↪
| trigger_uuid       | b85bdf42-d126-4223-9a64-8c10930447e3
↪
| id                 | 4a795b99-936d-4330-be4d-d2099b160075
↪
| name               | cherryppy
↪
+-----+
↪-----+
'cherryppy' workflows and their status:
+-----+-----+-----+
| wf_id | id                               | status                |
+-----+-----+-----+
| 1     | 97e7e2c1-8ba1-4320-9831-b5baef1d480d | DEPLOYMENT_COMPLETE |
+-----+-----+-----+

```

## 1.6 Connect to Your App

7. Connect to your app using the value in the `app_url` field.

```
$ curl <your_application_uri_here>
```

## 1.7 Update Your App

You can set up your Git repository to fire an `on_commit` action to make a webhook call to Solum each time you make a commit. The webhook call sends a POST request to [http://10.0.2.15:9777/v1/triggers/<trigger\\_id>](http://10.0.2.15:9777/v1/triggers/<trigger_id>) causing Solum to automatically build a new image and re-deploy your application.

To do this with a GitHub repo, go to your repo on the web, click on Settings, and then select "Webhooks & Services" from the left navigation menu. In the Webhooks section, click "Add Webhook", and enter your GitHub account password when prompted. Copy and paste the value of `trigger_uri` from your "solum app show" command into the "Payload URL" field. Note that this will only work if you have a public IP address or hostname in the `trigger_uri` field. Select the "application/vnd.github.v3+json" Payload version, determine if you only want to trigger this webhook on "git push" or if you want it for other events too by using the radio buttons and Checkboxes provided. Finish by clicking "Add Webhook". Now next time that event is triggered on GitHub, Solum will automatically check out your change, build it, and deploy it for you.

## 1.8 Languagepacks

Languagepacks define the runtime environment required by your application.

To build a languagepack, solum requires a git repo containing a Dockerfile. Solum creates a Docker and stores the image for use when building and deploying your application. See the sample languagepack repo below

```
$ https://github.com/rackspace-solum-samples/solum-languagepack-python
```

Here are some best practices to keep in mind while creating a languagepack

1. A good languagepack is reusable across application
2. All Operating system level libraries should be defined in the languagepack
3. Test tools should be installed in the languagepack
4. Includes a mandatory build.sh script, which Solum CI expects and executes during the build phase

## 1.9 appfile

An appfile is used to define your application and passed in during application creation.

```
$ solum app create --app-file appfile.yaml --param-file params.yaml
```

In the above command, we use the --app-file flag to provide

```
version: 1
name: cherrypy
description: python web app
languagepack: python
source:
  repository: https://github.com/rackspace-solum-samples/solum-python-
  ↪sample-app.git
  revision: master
workflow_config:
  test_cmd: ./unit_tests.sh
  run_cmd: python app.py
trigger_actions:
- test
- build
- deploy
ports:
- 80
```

The appfile is used to define the following

1. The git repo where your code exists
2. The languagepack to use
3. A name for your application
4. A command that executes your unittests. This command is executed during the unit test phase of the Solum CI workflow.

5. The port which is exposed publicly for accessing your application.
6. A command that executes your command.

### 1.10 App configuration and environment variables

Applications deployed using Solum can be configured using environment variables. Provide a parameter file during application creation to inject environment variables

```
$ solum app create --app-file appfile.yaml --param-file params.yaml
```

In the example above, we pass in the parameter file (shown in the table below) using the `--param-file` flag. The parameter file contains key value pairs which are injected into the application run time environment.

```
key: secret_key
user: user_name_goes_here
password: password_for_demo
```

### 1.11 Set up a Development Environment

These instructions are for those who want to contribute to Solum, or use features that are not yet in the latest release.

1. Clone the Solum repo. Solum repository is available on the OpenStack Git server.

```
$ mkdir ~/Solum
$ cd Solum
$ git clone https://github.com/openstack/solum.git
```

In addition to Solum, your environment will also need Devstack to configure and run the requisite OpenStack components, including Keystone, Glance, Nova, Neutron, and Heat.

### 1.12 Vagrant Dev Environment

2. We have provided a Vagrant environment to deploy Solum and its required OpenStack components via Devstack. We recommend using this approach if you are planning to contribute to Solum. This takes about the same amount of time as setting up Devstack manually, but it automates the setup for you. By default, it uses Virtualbox as its provisioner. We have tested this with Vagrant 1.5.4. The environment will need to know where your Solum code is, via the environment variable `SOLUM`.

```
$ cd ~/Solum
$ export SOLUM=~/.Solum/solum
$ git clone https://github.com/rackerlabs/vagrant-solum-dev.git vagrant
$ cd vagrant
```

3. Bring up the devstack vagrant environment. This may take a while. Allow about an hour, more or less depending on your machine speed and its connection to the internet.

```
$ vagrant up --provision devstack
$ vagrant ssh devstack
```

## 1.13 Devstack

Using Vagrant is not a requirement for deploying Solum. You may instead opt to install Solum and Devstack yourself. The details of integrating Solum with Devstack can be found in `devstack/README.rst`.





## INSTALL SOLUM

### 2.1 Distro specific installation

TODO add docs here on how to install on different distros like:

- debian
- redhat
- suse
- ubuntu

### 2.2 For a development installation use devstack



## ENABLING SOLUM IN DEVSTACK

1. Install Docker version 1.7.0 using following steps (Solum has been tested with this version of Docker):

```
echo deb http://get.docker.com/ubuntu docker main | sudo tee /etc/apt/  
↳sources.list.d/docker.list  
sudo apt-key adv --keyserver pgp.mit.edu --recv-keys_  
↳36A1D7869245C8950F966E92D8576A8BA88D21E9  
sudo apt-get update  
sudo apt-get install lxc-docker-1.7.0
```

2. Download DevStack:

```
git clone https://opendev.org/openstack/devstack.git  
cd devstack
```

3. Add this repo as an external repository:

```
cat > local.conf <<END  
[[local|localrc]]  
enable_plugin solum https://opendev.org/openstack/solum  
END
```

To use stable branches, make sure devstack is on that branch, and specify the branch name to `enable_plugin`, for example:

```
enable_plugin solum https://opendev.org/openstack/solum stable/mitaka
```

4. Run `./stack.sh`.

---

**Note:** This setup will produce virtual machines, not Docker containers. For an example of the Docker setup, see:

```
https://wiki.openstack.org/wiki/Solum/Docker
```

---



## CONFIGURE AND RUN SOLUM

### 4.1 Configuration Reference

To alter the default compute flavor edit `/etc/solum/templates/*.yaml`

```
flavor:  
  type: string  
  description: Flavor to use for servers  
  default: ml.tiny
```

Edit the default section to the desired value.

### 4.2 Administrator Guide

#### 4.2.1 Man pages for services and utilities

##### Solum utilities

##### **solum-db-manage**

##### **SYNOPSIS**

```
solum-db-manage <action> [options]
```

##### **DESCRIPTION**

solum-db-manage helps manage solum specific database operations.

The migrations in the "alembic\_migrations/versions/" directory contain the changes needed to migrate from older Solum releases to newer versions. A migration occurs by executing a script that details the changes needed to upgrade/downgrade the database. The migration scripts are ordered so that multiple scripts can run sequentially to update the database. The scripts are executed by Solum's migration wrapper which uses the Alembic library to manage the migration.

### OPTIONS

The standard pattern for executing a `solum-db-manage` command is:

```
solum-db-manage <command> [<args>]
```

Run with `-h` to see a list of available commands:

```
solum-db-manage -h
```

#### Commands are:

- `version`
- `upgrade`
- `downgrade`
- `stamp`
- `revision`

Detailed descriptions are below.

### Upgrading/Downgrading

If you are a deployer or developer and want to migrate from Icehouse to Juno or later you must first add version tracking to the database:

```
solum-db-manage stamp icehouse
```

You can then upgrade to the latest database version via:

```
solum-db-manage upgrade head
```

To check the current database version:

```
solum-db-manage version
```

Downgrade the database to a specific revision:

```
solum-db-manage downgrade 594288b1585a
```

### Generating migration templates (developers only)

A database migration script is required when you submit a change to Solum that alters the database model definition. The migration script is a special python file that includes code to update/downgrade the database to match the changes in the model definition. Alembic will execute these scripts in order to provide a linear migration path between revision. The `solum-db-manage` command can be used to generate migration template for you to complete. The operations in the template are those supported by the Alembic migration library.

```
solum-db-manage revision -m "description of revision" --autogenerate
```

This generates a prepopulated template with the changes needed to match the database state with the models. You should inspect the autogenerated template to ensure that the proper models have been altered.

In rare circumstances, you may want to start with an empty migration template and manually author the changes necessary for an upgrade/downgrade. You can create a blank file via:

```
solum-db-manage revision -m "description of revision"
```

## FILES

The `/etc/solum/solum.conf` file contains global options which can be used to configure some aspects of `solum-db-manage`, for example the DB connection and logging.

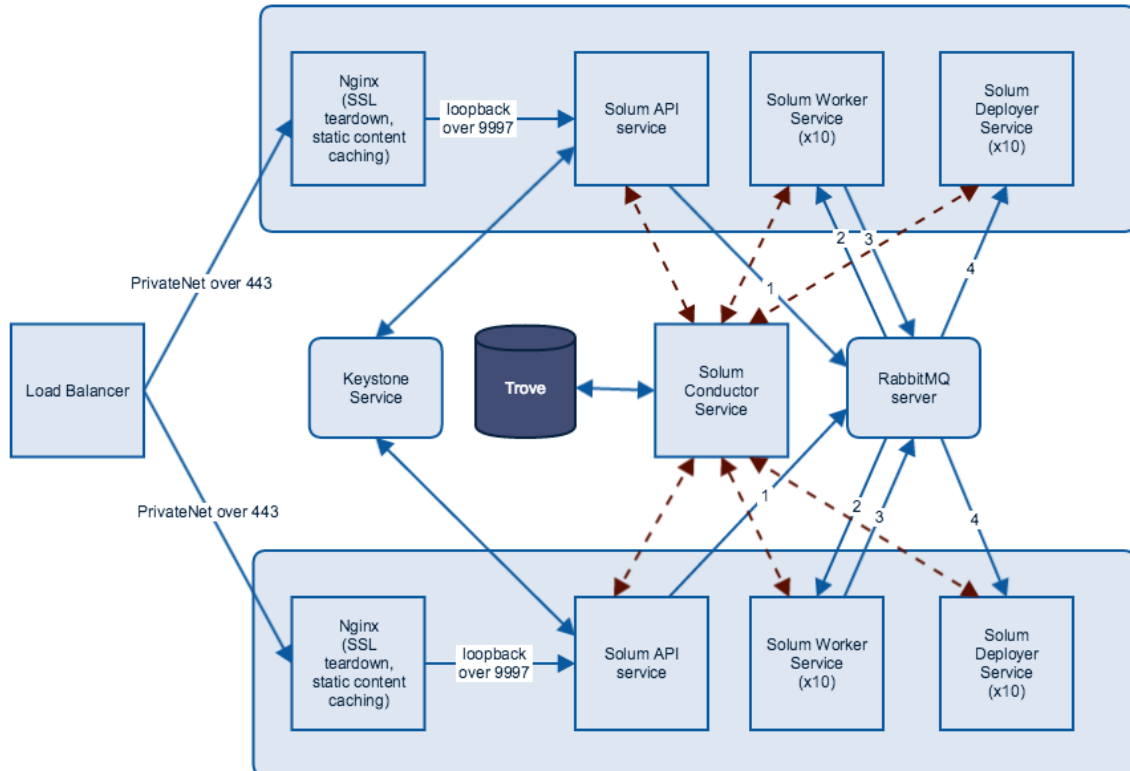
## BUGS

Solum issues are tracked in Launchpad so you can view or report bugs here: [OpenStack Bugs](#) [Solum Bugs](#)

## 4.3 High Availability Guide

## 4.4 Operations Guide

Solum has been successfully running in production environments with the following example architecture:



Solum application deployment follows this flow:

- Load Balancer listening on HTTPS port

- Traffic travels across private net to 2+ nodes to Nginx listening on port 443
- Nginx tears down SSL and redirects traffic over loopback to port 9777 to Solum API service
- Solum API Service authenticates with Keystone service (open up outbound traffic to only keystone service from Solum API)
- To retrieve Solum applications, API service would send messages to Conductor service, which communicates over service net to Trove to retrieve data
- During app deployment, Solum API service sends a queue message to Rabbit MQ service [1] (should be multi-node over private net)
- Solum Worker service picks up a queue message from Rabbit MQ [2] and pulls down a git repository, builds it, runs unit tests (if specified), builds a docker container, and uploads it to Swift \* This is a fairly lengthy process and completely blocks this service. You should scale out your infrastructure to easily accommodate your traffic. A performance test based on your expected load can give you a good idea of how many nodes and how many worker services per node you need.
- Solum Worker persists application state to Trove via Conductor service
- Upon completion, worker service sends a message to Rabbit MQ [3]
- Solum Deployer service picks up the message from Rabbit MQ [4] and calls Heat to deploy a heat stack with user's information and newly created docker container \* Deployer service also blocks on this call so your infrastructure should scale out to support your user load
- Deployer service persists application state to Trove via Conductor service

Solum deployment infrastructure is dependent on existence of the following OpenStack services:

- Nova
- Keystone
- Trove
- Swift
- Glance
- Heat

To assist with deploying a new Solum architecture, please refer to the following cookbooks to get started:

- <https://github.com/rackerlabs/cookbook-openstack-paas>
- <https://github.com/openstack/cookbook-openstack-identity.git>
- <https://github.com/openstack/cookbook-openstack-common.git>

## 4.5 Security Guide



## DEVELOP APPLICATIONS FOR SOLUM

### 5.1 API Complete Reference

#### 5.1.1 Version discovery

**type** `Version`

Version representation.

Data samples:

**Json**

```
{
  "id": "v1.0",
  "link": {
    "href": "http://example.com:9777/v1",
    "target_name": "v1"
  },
  "status": "CURRENT"
}
```

**XML**

```
b'<value>\n  <id>v1.0</id>\n  <status>CURRENT</status>\n  <link>\n↪n    <href>http://example.com:9777/v1</href>\n    <target_name>\n↪v1</target_name>\n  </link>\n</value>'
```

**id**

**Type** `str`

The version identifier.

**link**

**Type** `Link`

The link to the versioned API.

**status**

**Type** `Enum(SUPPORTED, CURRENT, DEPRECATED)`

The status of the API (SUPPORTED, CURRENT or DEPRECATED).

## 5.1.2 V1 API

### type Link

A link representation.

Data samples:

#### Json

```
{
  "href": "http://example.com:9777/v1",
  "target_name": "v1"
}
```

#### XML

```
b'<value>\n  <href>http://example.com:9777/v1</href>\n  <target_
↵name>v1</target_name>\n</value>'
```

#### href

**Type** str

The link URI.

#### target\_name

**Type** str

Textual name of the target link.

## Platform

### type Platform

Representation of a Platform.

The Platform resource is the root level resource that refers to all the other resources owned by this tenant.

Data samples:

#### Json

```
{
  "assemblies_uri": "http://example.com:9777/v1/assemblies",
  "components_uri": "http://example.com:9777/v1/components",
  "description": "solum native implementation",
  "extensions_uri": "http://example.com:9777/v1/extensions",
  "implementation_version": "2014.1.1",
  "infrastructure_uri": "http://example.com:9777/v1/
↵infrastructure",
  "language_packs_uri": "http://example.com:9777/v1/language_
↵packs",
  "name": "solum",
  "operations_uri": "http://example.com:9777/v1/operations",
  "pipelines_uri": "http://example.com:9777/v1/pipelines",
  "plans_uri": "http://example.com:9777/v1/plans",
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
}
```

(continues on next page)

(continued from previous page)

```

"sensors_uri": "http://example.com:9777/v1/sensors",
"services_uri": "http://example.com:9777/v1/services",
"tags": [
    "solid"
],
"triggers_uri": "http://example.com:9777/v1/triggers",
"type": "platform",
"uri": "http://example.com/v1",
"user_id": "55f41cf46df74320b9486a35f5d28a11"
}

```

## XML

```

b'<value>\n  <implementation_version>2014.1.1</implementation_
↪version>\n  <plans_uri>http://example.com:9777/v1/plans</plans_
↪uri>\n  <assemblies_uri>http://example.com:9777/v1/assemblies</
↪assemblies_uri>\n  <services_uri>http://example.com:9777/v1/
↪services</services_uri>\n  <components_uri>http://example.
↪com:9777/v1/components</components_uri>\n  <extensions_uri>
↪http://example.com:9777/v1/extensions</extensions_uri>\n
↪<operations_uri>http://example.com:9777/v1/operations</
↪operations_uri>\n  <sensors_uri>http://example.com:9777/v1/
↪sensors</sensors_uri>\n  <language_packs_uri>http://example.
↪com:9777/v1/language_packs</language_packs_uri>\n  <pipelines_
↪uri>http://example.com:9777/v1/pipelines</pipelines_uri>\n
↪<triggers_uri>http://example.com:9777/v1/triggers</triggers_uri>
↪\n  <infrastructure_uri>http://example.com:9777/v1/
↪infrastructure</infrastructure_uri>\n  <name>solum</name>\n
↪<type>platform</type>\n  <tags>\n    <item>solid</item>\n  </
↪tags>\n  <project_id>1dae5a09ef2b4d8cbf3594b0eb4f6b94</project_
↪id>\n  <user_id>55f41cf46df74320b9486a35f5d28a11</user_id>\n
↪<description>solum native implementation</description>\n  <uri>
↪http://example.com/v1</uri>\n</value>'

```

### **assemblies\_uri**

**Type** str

URI to assemblies.

### **components\_uri**

**Type** str

URI to components.

### **extensions\_uri**

**Type** str

URI to extensions.

### **implementation\_version**

**Type** str

Version of the platform.

### **infrastructure\_uri**

**Type** str

URI to infrastructure.

**language\_packs\_uri**

**Type** str

URI to language packs.

**operations\_uri**

**Type** str

URI to operations.

**pipelines\_uri**

**Type** str

URI to pipelines.

**plans\_uri**

**Type** str

URI to plans.

**sensors\_uri**

**Type** str

URI to sensors.

**services\_uri**

**Type** str

URI to services.

**triggers\_uri**

**Type** str

URI to triggers.

## Plans

**GET** /v1/plans

Return all plans, based on the query provided.

**POST** /v1/plans

Create a new plan.

**GET** /v1/plans/ (*plan\_id*)

Return this plan.

**PUT** /v1/plans/ (*plan\_id*)

Modify this plan.

**DELETE** /v1/plans/ (*plan\_id*)

Delete this plan.

**type Plan**

Representation of an Plan file.

The Plan resource is a representation of a Plan file. Plans are used to create Assembly resources. A Plan resource may be used to create an arbitrary number of Assembly instances. They use artifacts and services to indicate what will be used to generate the plan, and what services Solum can use to satisfy them. Note: Plan files are YAML and Plan resources are the REST representation of the Plan file after services have been matched to ones offered by Solum.

Data samples:

**Json**

```
{
  "artifacts": [
    {
      "artifact_type": "git_pull",
      "content": {
        "href": "git://example.com/project.git",
        "private": false
      },
      "language_pack": "65ed18ad-1d69-48bf-97c5-5c5945fd0c2d",
      "name": "My-python-app",
      "requirements": [
        {
          "fulfillment": "id:build",
          "requirement_type": "git_pull"
        }
      ]
    }
  ],
  "description": "A plan with no services or artifacts shown",
  "name": "Example-plan",
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "services": [
    {
      "characteristics": [
        "python_build_service"
      ],
      "id": "build",
      "name": "Build-Service"
    }
  ],
  "tags": [
    "small"
  ],
  "trigger_uri": "http://example.com/v1/triggers/1abc234",
  "type": "plan",
  "uri": "http://example.com/v1/plans/x1",
  "user_id": "55f41cf46df74320b9486a35f5d28a11"
}
```

**XML**

```
b'<value>\n  <artifacts>\n    <item>\n      <name>My-python-app/</<br>
<name>\n      <artifact_type>git_pull</artifact_type>\n<br>
<content>\n      <item>\n        <key>href</key>\n        <value>git://example.com/project.git</value>\n        <key>private</key>\n        <value>false</value>\n      </item>\n    </content>\n    <key>language_pack</key>\n    <value>65ed18ad-1d69-48bf-97c5-5c5945fd0c2d</value>\n  </item>\n  <key>name</key>\n  <value>Example-plan</value>\n  <key>project_id</key>\n  <value>1dae5a09ef2b4d8cbf3594b0eb4f6b94</value>\n  <key>services</key>\n  <value>[<br>
    <item>\n      <key>characteristics</key>\n      <value>[<br>
        <item>\n          <value>python_build_service</value>\n        </item>\n      </value>\n      <key>id</key>\n      <value>build</value>\n      <key>name</key>\n      <value>Build-Service</value>\n    </item>\n  </value>\n  <key>tags</key>\n  <value>[<br>
    <item>\n      <value>small</value>\n    </item>\n  </value>\n  <key>trigger_uri</key>\n  <value>http://example.com/v1/triggers/1abc234</value>\n  <key>type</key>\n  <value>plan</value>\n  <key>uri</key>\n  <value>http://example.com/v1/plans/x1</value>\n  <key>user_id</key>\n  <value>55f41cf46df74320b9486a35f5d28a11</value>\n</value>'
```

---

**artifacts**

**Type** list(Artifact)

List of artifacts defining the plan.

**parameters**

**Type** dict(str: None)

User defined parameters

**services**

**Type** list(ServiceReference)

List of services needed by the plan.

**trigger\_uri**

**Type** str

The trigger uri used to trigger the build of the plan

## Pipelines

**GET** /v1/pipelines

Return all pipelines.

**Return type** list(*Pipeline*)

**POST** /v1/pipelines

Create a new pipeline. :type data: *Pipeline*

**Return type** *Pipeline*

**GET** /v1/pipelines/ (*pipeline\_id*)

Return this pipeline.

**Return type** *Pipeline*

**PUT** /v1/pipelines/ (*pipeline\_id*)

Modify this pipeline. :type data: *Pipeline*

**Return type** *Pipeline*

**DELETE** /v1/pipelines/ (*pipeline\_id*)

Delete this pipeline.

**type Pipeline**

Representation of an Pipeline.

A pipeline is the association between a plan, a mistral workbook and a git trigger. Together they form a working development "pipeline".

Data samples:

**Json**

```
{
  "description": "A pipeline for my app",
  "last_execution": "78f41cf46df7430b9486a35f5d28a41",
  "name": "Example-pipeline",
  "plan_uri": "http://example.com/v1/plans/x1",
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "tags": [
    "small"
  ],
  "trigger_uri": "http://example.com/v1/triggers/1abc234",
  "type": "pipeline",
  "uri": "http://example.com/v2/pipelines/p1",
  "user_id": "55f41cf46df74320b9486a35f5d28a11",
  "workbook_name": "build-deploy"
}
```

## XML

```
b'<value>\n  <plan_uri>http://example.com/v1/plans/x1</plan_uri>\n
↪\n  <workbook_name>build-deploy</workbook_name>\n  <trigger_uri>
↪http://example.com/v1/triggers/1abc234</trigger_uri>\n  <last_
↪execution>78f41cf46df7430b9486a35f5d28a41</last_execution>\n
↪<type>pipeline</type>\n  <name>Example-pipeline</name>\n
↪<description>A pipeline for my app</description>\n  <tags>\n
↪<item>small</item>\n  </tags>\n  <project_id>
↪1dae5a09ef2b4d8cbf3594b0eb4f6b94</project_id>\n  <user_id>
↪55f41cf46df74320b9486a35f5d28a11</user_id>\n  <uri>http://
↪example.com/v2/pipelines/p1</uri>\n</value>'
```

### **last\_execution**

**Type** str

The UUID of the last run execution.

### **plan\_uri**

**Type** str

Link to the plan URI.

### **trigger\_uri**

**Type** str

The trigger uri used to trigger the pipeline.

### **workbook\_name**

**Type** str

Name of the workbook in Mistral to use.

## Executions

**GET** `/v1/pipelines/ (pipeline_id) /executions`

Return all executions, based on the provided pipeline\_id. :type pipeline\_id: str

**Return type** list(*Execution*)

**type** *Execution*

Data samples:

**Json**

```
{
  "href": "http://example.com:9777/v1",
  "target_name": "v1"
}
```

**XML**

```
b'<value>\n  <href>http://example.com:9777/v1</href>\n  <target_
↵name>v1</target_name>\n</value>'
```

## Assemblies

**GET** `/v1/assemblies`

Return all assemblies, based on the query provided.

**Return type** list(*Assembly*)

**POST** `/v1/assemblies`

Create a new assembly. :type data: *Assembly*

**Return type** *Assembly*

**GET** `/v1/assemblies/ (assembly_id)`

Return this assembly.

**Return type** *Assembly*

**PUT** `/v1/assemblies/ (assembly_id)`

Modify this assembly. :type data: *Assembly*

**Return type** *Assembly*

**DELETE** `/v1/assemblies/ (assembly_id)`

Delete this assembly.

**type** *Assembly*

Representation of an Assembly.

The Assembly resource represents a group of components that make up a running instance of an application. You may casually refer to this as "the application" but we refer to it as an Assembly because most cloud applications are actually a system of multiple service instances that make up a system. For example, a three-tier web application may have a load balancer component, a group of application servers, and a database server all represented as Component resources that make up an Assembly resource. An Assembly resource has at least one Component resource associated with it.



Data samples:

### Json

```
{
  "components": [],
  "created_at": "2021-04-06T06:51:28.870217",
  "description": "A mysql database",
  "name": "database",
  "operations": [],
  "plan_uri": "http://example.com/v1/plans/45-09",
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "sensors": [],
  "tags": [
    "small"
  ],
  "type": "assembly",
  "updated_at": "2021-04-06T06:51:28.870217",
  "uri": "http://example.com/v1/assemblies/x4",
  "user_id": "55f41cf46df74320b9486a35f5d28a11"
}
```

### XML

```
b'<value>\n  <plan_uri>http://example.com/v1/plans/45-09</plan_
↪uri>\n  <components />\n  <operations />\n  <sensors />\n
↪<updated_at>2021-04-06T06:51:28.870217</updated_at>\n  <created_
↪at>2021-04-06T06:51:28.870217</created_at>\n  <name>database</
↪name>\n  <type>assembly</type>\n  <tags>\n    <item>small</item>
↪\n  </tags>\n  <project_id>1dae5a09ef2b4d8cbf3594b0eb4f6b94</
↪project_id>\n  <user_id>55f41cf46df74320b9486a35f5d28a11</user_
↪id>\n  <description>A mysql database</description>\n  <uri>
↪http://example.com/v1/assemblies/x4</uri>\n</value>'
```

#### **application\_uri**

**Type** str

The uri of the deployed application.

#### **components**

**Type** list(Component)

Components that belong to the assembly.

#### **created\_at**

**Type** datetime

The time the assembly initially created.

#### **operations**

**Type** list(Operation)

Operations that belong to the assembly.

#### **plan\_uri**

**Type** str

The URI to the plan to be used to create this assembly.

**sensors**

**Type** list(Sensor)

Sensors that belong to the assembly.

**status**

**Type** str

The status of the assembly.

**updated\_at**

**Type** datetime

The last time a change was made to the assembly's status.

**workflow**

**Type** list(Enum(unittest, build, deploy))

Defines the workflow that an assembly will go through.

## Services

**GET /v1/services**

Return all services, based on the query provided.

**Return type** list(*Service*)

**POST /v1/services**

Create a new service. :type data: *Service*

**Return type** *Service*

**GET /v1/services/ (*service\_id*)**

Return this service.

**Return type** *Service*

**PUT /v1/services/ (*service\_id*)**

Modify this service. :type data: *Service*

**Return type** *Service*

**DELETE /v1/services/ (*service\_id*)**

Delete this service.

**type Service**

The Service resource represents a networked service.

You may create Component resources that refer to Service resources. A Component represents an instance of a Service. Your application connects to such a Component using a network protocol. For example, the Platform may offer a default Service named "mysql". You may create multiple Component resources that reference different instances of the "mysql" service. Each Component may be a multi-tenant instance of a MySQL database (perhaps a logical database) service offered by the Platform for a given Assembly.

Data samples:

## Json

```
{
  "description": "A language pack service",
  "name": "language-pack",
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "read_only": false,
  "service_type": "language_pack",
  "tags": [
    "group_xyz"
  ],
  "type": "service",
  "uri": "http://example.com/v1/language_packs/java1.4",
  "user_id": "55f41cf46df74320b9486a35f5d28a11"
}
```

## XML

```
b'<value>\n  <read_only>>false</read_only>\n  <service_type>
↪ language_pack</service_type>\n  <name>language-pack</name>\n
↪ <type>service</type>\n  <description>A language pack service</
↪ description>\n  <project_id>1dae5a09ef2b4d8cbf3594b0eb4f6b94</
↪ project_id>\n  <user_id>55f41cf46df74320b9486a35f5d28a11</user_
↪ id>\n  <tags>\n    <item>group_xyz</item>\n  </tags>\n  <uri>
↪ http://example.com/v1/language_packs/java1.4</uri>\n</value>'
```

### read\_only

**Type** bool

The service is read only when this value is true.

### service\_type

**Type** str

Type of service. Example: language\_pack or db::mysql

## Operations

### GET /v1/operations

Return all operations, based on the query provided.

**Return type** list(*Operation*)

### POST /v1/operations

Create a new operation. :type data: *Operation*

**Return type** *Operation*

### GET /v1/operations/ (*operation\_id*)

Return this operation.

**Return type** *Operation*

### PUT /v1/operations/ (*operation\_id*)

Modify this operation. :type data: str

**Return type** *Operation*

**DELETE** /v1/operations/ (*operation\_id*)

Delete this operation.

### type Operation

An Operation resource represents an operation or action.

This is for defining actions that may change the state of the resource they are related to. For example, the API already provides ways to register, start, and stop your application (POST an Assembly to register+start, and DELETE an Assembly to stop) but Operations provide a way to extend the system to add your own actions such as "pause" and "resume", or "scale\_up" and "scale\_down".

Data samples:

### Json

```
{
  "description": "A resume operation",
  "documentation": "http://example.com/docs/resume_op",
  "name": "resume",
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "tags": [
    "small"
  ],
  "target_resource": "http://example.com/instances/uuid",
  "type": "operation",
  "uri": "http://example.com/v1/operations/resume",
  "user_id": "55f41cf46df74320b9486a35f5d28a11"
}
```

### XML

```
b'<value>\n  <documentation>http://example.com/docs/resume_op</
  <documentation>\n  <target_resource>http://example.com/instances/
  <target_resource>\n  <name>resume</name>\n  <type>operation
  </type>\n  <tags>\n    <item>small</item>\n  </tags>\n
  <project_id>1dae5a09ef2b4d8cbf3594b0eb4f6b94</project_id>\n
  <user_id>55f41cf46df74320b9486a35f5d28a11</user_id>\n
  <description>A resume operation</description>\n  <uri>http://
  <uri>\n</value>'
```

### documentation

Type str

Documentation URI for the operation.

### target\_resource

Type str

Target resource URI to the operation.

## Sensors

### GET /v1/sensors

Return all sensors, based on the query provided.

**Return type** `list(Sensor)`

### POST /v1/sensors

Create a new sensor. :type data: str

**Return type** `Sensor`

### GET /v1/sensors/ (sensor\_id)

Return this sensor.

**Return type** `Sensor`

### PUT /v1/sensors/ (sensor\_id)

Modify this sensor. :type data: str

**Return type** `Sensor`

### DELETE /v1/sensors/ (sensor\_id)

Delete this sensor.

### type Sensor

A Sensor resource represents exactly one supported sensor.

Sensor resources represent dynamic data about resources, such as metrics or state. Sensor resources are useful for exposing data that changes rapidly, or that may need to be fetched from a secondary system.

Data samples:

### Json

```
{
  "description": "A heartbeat sensor",
  "documentation": "http://example.com/docs/heartbeat/",
  "name": "hb",
  "operations": [],
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "sensor_type": "str",
  "target_resource": "http://example.com/instances/uuid",
  "timestamp": "2021-04-06T06:51:29.009559",
  "type": "sensor",
  "uri": "http://example.com/v1/sensors/hb",
  "user_id": "55f41cf46df74320b9486a35f5d28a11",
  "value": "30"
}
```

### XML

```
b'<value>\n  <documentation>http://example.com/docs/heartbeat/</
↪documentation>\n  <target_resource>http://example.com/instances/
↪uuid</target_resource>\n  <sensor_type>str</sensor_type>\n
↪<timestamp>2021-04-06T06:51:29.009559</timestamp>\n
↪<operations />\n  <value>30</value>\n  <name>hb</name>\n  <type>
↪sensor</type>\n  <project_id>1dae5a09ef2b4d8cbf3594b0eb4f6b94</
↪project_id>\n  <user_id>55f41cf46df74320b9486a35f5d28a11</user
↪id>\n  <description>A heartbeat sensor</description>\n  <uri>
↪http://example.com/v1/sensors/hb</uri>\n</value>'
```

(continues on next page)

---

**documentation**

**Type** str

Documentation URI for the sensor.

**operations**

**Type** list(Operation)

Operations that belong to the sensor.

**sensor\_type**

**Type** Enum(str, float, int)

Sensor data type.

**target\_resource**

**Type** str

Target resource URI to the sensor.

**timestamp**

**Type** datetime

Timestamp for Sensor.

**property value**

Value of the sensor.

## Components

**GET /v1/components**

Return all components, based on the query provided.

**Return type** list(*Component*)

**POST /v1/components**

Create a new component. :type data: *Component*

**Return type** *Component*

**GET /v1/components/ (component\_id)**

Return this component.

**Return type** *Component*

**PUT /v1/components/ (component\_id)**

Modify this component. :type data: *Component*

**Return type** *Component*

**DELETE /v1/components/ (component\_id)**

Delete this component.

**type Component**

The Component resource represents one part of an Assembly.

For example, an instance of a database service may be a Component. A Component resource may also represent a static artifact, such as an archive file that contains data for initializing your application. An Assembly may have different components that represent different processes that run. For example, you may have one Component that represents an API service process, and another that represents a web UI process that consumes that API service. The simplest case is when an Assembly has only one component. For example, your component may be named "PHP" and refers to the PHP Service offered by the platform for running a PHP application.

Data samples:

**Json**

```
{
  "abbreviated": true,
  "component_type": "heat_stack",
  "components_ids": [],
  "description": "A php web application component",
  "heat_stack_id": "4c712026-dcd5-4664-90b8-0915494c1332",
  "name": "php-web-app",
  "operations": [],
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "sensors": [],
  "services": [],
  "tags": [
    "group_xyz"
  ],
  "type": "component",
  "uri": "http://example.com/v1/components/php-web-app",
  "user_id": "55f41cf46df74320b9486a35f5d28a11"
}
```

**XML**

```
b'<value>\n  <services />\n  <operations />\n  <sensors />\n
↳<abbreviated>>true</abbreviated>\n  <components_ids />\n
↳<component_type>heat_stack</component_type>\n  <heat_stack_id>
↳4c712026-dcd5-4664-90b8-0915494c1332</heat_stack_id>\n  <name>
↳php-web-app</name>\n  <type>component</type>\n  <description>A_
↳php web application component</description>\n  <tags>\n
↳<item>group_xyz</item>\n  </tags>\n  <project_id>
↳1dae5a09ef2b4d8cbf3594b0eb4f6b94</project_id>\n  <user_id>
↳55f41cf46df74320b9486a35f5d28a11</user_id>\n  <uri>http://
↳example.com/v1/components/php-web-app</uri>\n</value>'
```

**abbreviated**

**Type** bool

Boolean value indicating if this components has nested components at more than one level of depth.

**assembly\_uuid**

**Type** str

"The uuid of the assembly that this component belongs in.

**component\_type**

**Type** str

Type of component e.g. heat\_stack.

**components\_ids**

**Type** list(str)

IDs of nested component of the component.

**heat\_stack\_id**

**Type** str

Unique identifier of the Heat Stack.

**operations**

**Type** list(Operation)

Operations that belong to the component.

**plan\_uri**

**Type** str

URI of Plan of which the component is a part.

**resource\_uri**

**Type** str

Remote resource URI of the component.

**sensors**

**Type** list(Sensor)

Sensors that belong to the component.

**services**

**Type** list(Service)

Services that belong to the component.

## Extensions

**GET /v1/extensions**

Return all extensions, based on the query provided.

**Return type** list(*Extension*)

**POST /v1/extensions**

Create a new extension. :type data: str

**Return type** *Extension*

**GET /v1/extensions/ (*extension\_id*)**

Return this extension.

**Return type** *Extension*



**PUT** `/v1/extensions/` (*extension\_id*)  
 Modify this extension. :type data: str

**Return type** *Extension*

**DELETE** `/v1/extensions/` (*extension\_id*)  
 Delete this extension.

### type **Extension**

The Extension resource represents Provider modifications.

This may include additional protocol semantics, resource types, application lifecycle states, resource attributes, etc. Anything may be added, as long as it does not contradict the base functionality offered by Solum.

Data samples:

#### Json

```
{
  "description": "This logstash extension provides a tool for
  ↪managing your application events and logs.",
  "documentation": "http://example.com/docs/ext/logstash",
  "name": "logstash",
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "tags": [
    "large"
  ],
  "type": "extension",
  "uri": "http://example.com/v1/extensions/logstash",
  "user_id": "55f41cf46df74320b9486a35f5d28a11",
  "version": "2.13"
}
```

#### XML

```
b'<value>\n  <version>2.13</version>\n  <documentation>http://
  ↪example.com/docs/ext/logstash</documentation>\n  <name>logstash
  ↪</name>\n  <type>extension</type>\n  <tags>\n    <item>large</
  ↪item>\n  </tags>\n  <project_id>1dae5a09ef2b4d8cbf3594b0eb4f6b94
  ↪</project_id>\n  <user_id>55f41cf46df74320b9486a35f5d28a11</
  ↪user_id>\n  <description>This logstash extension provides a
  ↪tool for managing your application events and logs.</
  ↪description>\n  <uri>http://example.com/v1/extensions/logstash</
  ↪uri>\n</value>'
```

#### **documentation**

**Type** str

Documentation URI to the extension.

#### **version**

**Type** str

Version of the extension.

## LanguagePacks

### GET /v1/language\_packs

Return all languagepacks, based on the query provided.

**Return type** `list(LanguagePack)`

### POST /v1/language\_packs

Create a new languagepack. :type data: `LanguagePack`

**Return type** `LanguagePack`

### GET /v1/language\_packs/ (lp\_id)

Return a languagepack.

**Return type** `LanguagePack`

### DELETE /v1/language\_packs/ (lp\_id)

Delete a languagepack.

### type LanguagePack

Representation of a language pack.

When a user creates an application, he specifies the language pack to be used. The language pack is responsible for building the application and producing an artifact for deployment.

For a complete list of language pack attributes please refer: <https://etherpad.openstack.org/p/Solum-Language-pack-json-format>

Data samples:

### Json

```
{
  "attributes": {
    "admin_email": "someadmin@somewhere.com",
    "optional_attr1": "value"
  },
  "base_image_id": "4dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "build_tool_chain": [
    {
      "type": "ant",
      "version": "1.7"
    },
    {
      "type": "maven",
      "version": "1.2"
    }
  ],
  "compiler_versions": [
    "1.4",
    "1.6",
    "1.7"
  ],
  "created_image_id": "4afasa09ef2b4d8cbf3594b0ec4f6b94",
  "description": "A php web application",
  "image_format": "docker",
  "language_implementation": "Sun",
  "language_pack_type": "org.openstack.solum.Java",

```

(continues on next page)

(continued from previous page)

```

"name": "php-web-app",
"os_platform": {
  "OS": "Ubuntu",
  "version": "12.04"
},
"project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
"runtime_versions": [
  "1.4",
  "1.6",
  "1.7"
],
"source_format": "heroku",
"source_uri": "git://example.com/project/app.git",
"tags": [
  "group_xyz"
],
"type": "languagepack",
"uri": "http://example.com/v1/images/b3e0d79",
"user_id": "55f41cf46df74320b9486a35f5d28a11"
}

```

## XML

```

b'<value>\n  <name>php-web-app</name>\n  <language_pack_type>org.
↳openstack.solum.Java</language_pack_type>\n  <compiler_versions>
↳\n    <item>1.4</item>\n    <item>1.6</item>\n    <item>1.7</
↳item>\n  </compiler_versions>\n  <runtime_versions>\n    <item>
↳1.4</item>\n    <item>1.6</item>\n    <item>1.7</item>\n  </
↳runtime_versions>\n  <language_implementation>Sun</language_
↳implementation>\n  <build_tool_chain>\n    <item>\n      <type>
↳ant</type>\n      <version>1.7</version>\n    </item>\n
↳<item>\n      <type>maven</type>\n      <version>1.2</version>\n
↳\n    </item>\n  </build_tool_chain>\n  <os_platform>\n    <item>
↳\n      <key>OS</key>\n      <value>Ubuntu</value>\n    </item>\n
↳\n    <item>\n      <key>version</key>\n      <value>12.04</
↳value>\n    </item>\n  </os_platform>\n  <attributes>\n
↳<item>\n      <key>optional_attr1</key>\n      <value>value</
↳value>\n    </item>\n    <item>\n      <key>admin_email</key>\n
↳\n      <value>someadmin@somewhere.com</value>\n    </item>\n  </
↳attributes>\n  <source_uri>git://example.com/project/app.git</
↳source_uri>\n  <source_format>heroku</source_format>\n  <base_
↳image_id>4dae5a09ef2b4d8cbf3594b0eb4f6b94</base_image_id>\n
↳<image_format>docker</image_format>\n  <created_image_id>
↳4afasa09ef2b4d8cbf3594b0ec4f6b94</created_image_id>\n  <tags>\n
↳\n    <item>group_xyz</item>\n  </tags>\n  <type>languagepack</
↳type>\n  <description>A php web application</description>\n
↳<project_id>1dae5a09ef2b4d8cbf3594b0eb4f6b94</project_id>\n
↳<user_id>55f41cf46df74320b9486a35f5d28a11</user_id>\n  <uri>
↳http://example.com/v1/images/b3e0d79</uri>\n</value>'

```

## attributes

Type dict(str: str)

Additional section attributes will be used to expose custom attributes designed by language pack creator.

**base\_image\_id**

**Type** str

The id (in glance) of the image to customize.

**build\_tool\_chain**

**Type** list(BuildTool)

Toolchain available in the language pack. Example: For a java language pack which supports Ant and Maven, build\_tool\_chain = [{"type:ant,version:1.7"},"{"type:maven,version:1.2}"]

**compiler\_versions**

**Type** list(str)

List of all the compiler versions supported by the language pack. Example: For a java language pack supporting Java versions 1.4 to 1.7, version = ['1.4', '1.6', '1.7']

**created\_image\_id**

**Type** str

The id of the created image in glance.

**image\_format**

**Type** Enum(auto, qcow2, docker)

The image format.

**language\_implementation**

**Type** str

Actual language implementation supported by the language pack. Example: In case of java it might be 'Sun' or 'openJava' In case of C++ it can be 'gcc' or 'icc' or 'microsoft'.

**language\_pack\_type**

**Type** str

Type of the language pack. Identifies the language supported by the language pack. This attribute value will use the org.openstack.solum namespace.

**lp\_metadata**

**Type** str

The languagepack meta data.

**os\_platform**

**Type** dict(str: str)

OS and its version used by the language pack. This attribute identifies the base image of the language pack.

**runtime\_versions**

**Type** list(str)

List of all runtime versions supported by the language pack. Runtime version can be different than compiler version. Example: An application can be compiled with java 1.7 but it should run in java 1.6 as it is backward compatible.

**source\_format****Type** Enum(auto, heroku, dib, dockerfile)

The source repository format.

**source\_uri****Type** str

The URI of the app/element.

**status****Type** Enum(QUEUED, BUILDING, ERROR, READY)

The state of the image.

## Infrastructure

**type Infrastructure**

Description of an Infrastructure.

Data samples:

**Json**

```
{
  "description": "Solum Infrastructure endpoint",
  "name": "infrastructure",
  "project_id": "1dae5a09ef2b4d8cbf3594b0eb4f6b94",
  "stacks_uri": "http://example.com/v1/infrastructure/stacks",
  "tags": [
    "small"
  ],
  "type": "infrastructure",
  "uri": "http://example.com/v1/infrastructure",
  "user_id": "55f41cf46df74320b9486a35f5d28a11"
}
```

**XML**

```
b'<value>\n  <stacks_uri>http://example.com/v1/infrastructure/
↪stacks</stacks_uri>\n  <name>infrastructure</name>\n  <type>
↪infrastructure</type>\n  <tags>\n    <item>small</item>\n  </
↪tags>\n  <project_id>1dae5a09ef2b4d8cbf3594b0eb4f6b94</project_
↪id>\n  <user_id>55f41cf46df74320b9486a35f5d28a11</user_id>\n
↪<description>Solum Infrastructure endpoint</description>\n
↪<uri>http://example.com/v1/infrastructure</uri>\n</value>'
```

**stacks\_uri****Type** str

URI to services.

## Triggers

### **POST /v1/triggers**

Trigger a new event on Solum.

## HOW TO CONTRIBUTE TO SOLUM

**If you would like to contribute to Solum, please see our contributing wiki:** <https://wiki.openstack.org/wiki/Solum/Contributing>

We have the same CLA requirements as OpenStack, so you must follow the steps in the "If you're a developer, start here" section of this page:

<https://docs.openstack.org/infra/manual/developers.html>

Once those steps have been completed, submit your changes to for review via the Gerrit tool, following the workflow documented at:

<https://docs.openstack.org/infra/manual/developers.html#development-workflow>

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad, not GitHub:

<https://bugs.launchpad.net/solum>

For tips to help with running unit tests and functional tests on your code, see:

<https://wiki.openstack.org/wiki/Solum/Testing>





## 7.1 solum-status

### 7.1.1 Synopsis

```
solum-status <category> <command> [<args>]
```

### 7.1.2 Description

**solum-status** is a tool that provides routines for checking the status of a Solum deployment.

### 7.1.3 Options

The standard pattern for executing a **solum-status** command is:

```
solum-status <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
solum-status
```

Categories are:

- upgrade

Detailed descriptions are below.

You can also run with a category argument such as `upgrade` to see a list of all commands in that category:

```
solum-status upgrade
```

These sections describe the available categories and arguments for **solum-status**.

## Upgrade

**solum-status upgrade check** Performs a release-specific readiness check before restarting services with new code. This command expects to have complete configuration and access to databases and services.

### Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

### History of Checks

#### 5.8.0 (Stein)

- Placeholder to be filled in with checks as they are added in Stein.