cronus Documentation

Release 0.1.45

binyu

January 08, 2015

Contents

CHAPTER 1

Getting Started

1.1 Quick Setup

Steps to install Cronus agent and Cronus master on a machine.

1.1.1 Prerequisites

- linux (Ubuntu, Cent, Redhat, Fedora etc.)
- sudo permission
- python > 2.6.7 && < 3
- openssl > 0.9.8
- wget
- system management daemon systemd or upstart
- unzip (for CronusMaster only)
- java runtime >= 1.5 (for CronusMaster only)

1.1.2 Install Agent and Master

```
cd ~; wget -qO- 'http://cronuspaas.github.io/downloads/install_cronusmaster' | bash
```

Expected outcome:

- CronusAgent is running at https://host:12020/agent
- CronusMaster is running at http://host:9000
- CronusMaster code is saved in ~/cronusmaster-master

Deploy First App

Now you can try to deploy the first app to local machine through cronusmaster. In CronusMaster (http://host:9000), run "Deploy Helloworld App", wait for it complete, check sample application running at http://host:8999

1.1.3 Install Agent Only

```
cd /tmp; wget -qO- 'http://cronuspaas.github.io/downloads/install_agent' | sudo dev=true bash
```

expected outcome:

- CronusAgent is running at https://localhost:12020/agent

1.1.4 Tested IaaS Compute Profiles

The following IaaS compute profiles have been tested working with Cronus.

AWS

- Ubuntu12.04
- Ubuntu14.04
- RHEL7

GCP

- Ubuntu12.04
- Ubuntu14.04
- RHEL7
- CentOS7

Azure

- Ubuntu12.04
- Ubuntu14.04
- Openlogic7

1.2 Advance Setup

1.2.1 Advance CronusMaster Settings

Default cronusmaster installation uses application.conf, edit configurations of application.conf.{env} for additional features

```
# cronusmaster public and private IP, needed for smart parameters in command
agentmaster.externalIp=cronusmaster_external_ip
agentmaster.internalIp=cronusmaster_internal_ip

# user data location (applicable only if use local FS for user data)
agentmaster.userDataDao.file.dir=.appdata

# agent basic authentication, need this if remote agent auth is enabled
agentmaster.cronusagent.password=username:password
```

Redeploy CronusMaster after changes `~/cronusmaster-master/scripts/install.sh {env}`

1.2.2 Advance CronusAgent Settings

Default cronusagent installation uses default SSL cert, and no authentication, install agent with the following parameters for additional features

```
# provide path_to_ssl_cert for custom ssl cert
# set user:password for basic authentication
wget -qO- 'http://cronuspaas.github.io/downloads/install_agent' \
| sudo server_pem=path_to_ssl_cert agent_pwd=user:password bash
```

Agent is reployed with new ssl cert, and basic authentication enabled, once authentication enabled, CronusMaster configuration of "agentmaster.cronusagent.password" needs to be updated to be able to work with updated agent.

1.3 Deploy Application Workflow

This document describe steps to deploy an application to cloud by using Cronus framework.

1.3.1 Prerequisites

- Cronus agent and Cronus master are already installed

1.3.2 Application Packaging

- Checkout desired application, package and upload

  ```
  cd ~
  git clone https://github.com/cronuspaas/cronusstacks.git
  cd cronusstacks/node_app
  ./package.sh upload
  ```

  Expected outcome: node_app is packaged and uploaded to local CronusMaster, check http://localhost:9000/agent/packages for the uploaded package

1.3.3 Deploy and Rollback

- Deploy
  1. In CronusMaster, navigate to command -> oneclick launch
  2. Run "deploy_nodeapp_local", wait for it to complete
  3. Check for nodeapp at http://localhost:1337/
- Rollback
  1. You will need to deploy another version before rollback, make some change to the nodeapp code, repeat above step for packaging, upload, deploy, and verify that your change is deployed
  2. Run oneclick launch "rollback_nodeapp_local", wait for it to complete
  3. Check the change is rolled back

1.3.4 Startup, Shutdown and Restart

- In CronusMaster, navigate to commands -> commands
- Run Agent_Service_LCMAction, go through command wizard, select nodegroup “Localhost”, fill in json user data of “serviceName”: “nodeapp”, “action”: “restart”, execute
- Wait for job to complete, click fulltext search in cmd job page to see the raw script output, note that shutdown and startup scripts are called respectively
- Repeat the above step for action=”shutdown”, and action=”startup”

1.3.5 Other Excercises

- Create a new nodegroup with list of nodes that are not localhost, install agent on them and deploy the same node application to the new nodegroup
- Create a recurring job that deploy latest nodeapp to localhost every 5 minutes
- Deploy a tomcat application to localhost, or create a new application stack ready for cronus deployment from scratch

CHAPTER 2

Cronus Master

2.1 Introduction

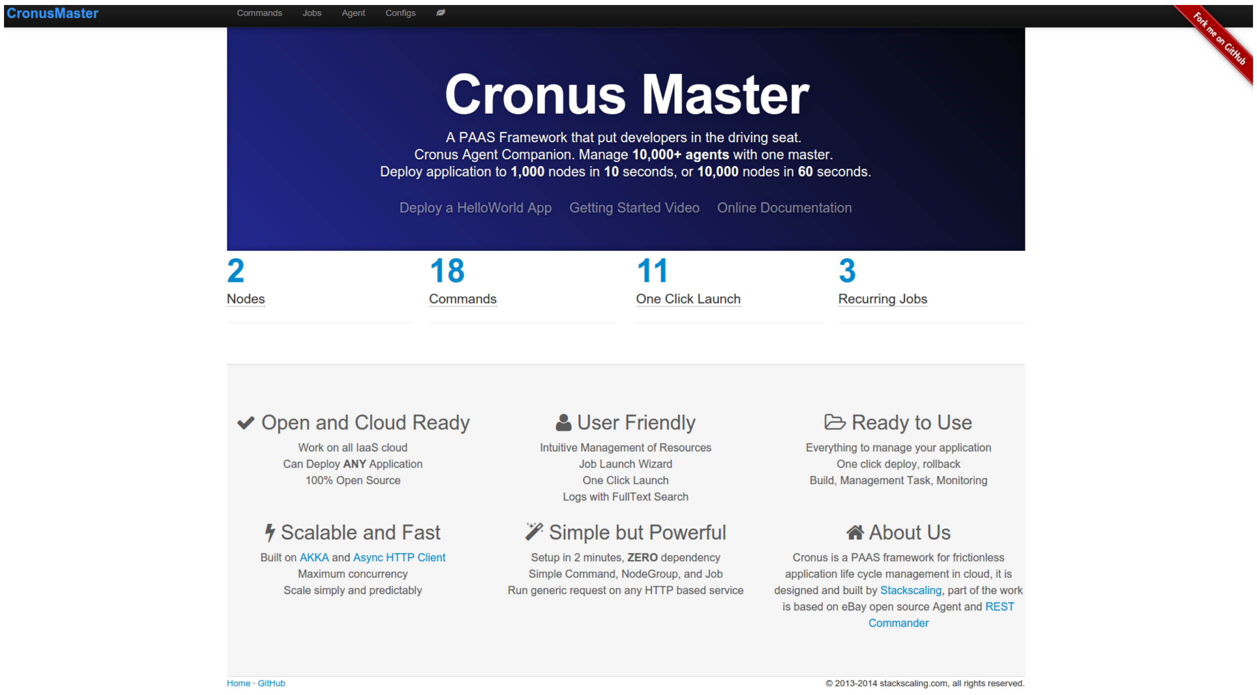
Cronusmaster can execute any http(s) request against one or many target nodes. Think of it as a RESTful client that can run with predefined template on target node grammatically at scale. Cronusmaster is used to execute different tasks on cronus agent through its RESTful interface.

- Command: Defines a HTTP(S) request template
- Nodegroup: Defines a list of nodes that will be targets of a command
- Job: A command that runs on a nodegroup
- Scheduled Job: A command run on a nodegroup with a recurring schedule
- Log: Execution result and log of a job

To put cronusmaster to work

1. Define HTTP(S) command template
2. Define nodegroup with target node IPs or hostnames
3. Execute materialized commmand on a nodegroup as a job
4. Check result of the job in the log
5. Optinoally create an oneclick launch from an already executed job
6. Optionally create a recurring job from an already executed job or oneclick launch

cronusmaster main page



Cronus Master

A PAAS Framework that put developers in the driving seat.
Cronus Agent Companion. Manage **10,000+ agents** with one master.
Deploy application to **1,000** nodes in **10** seconds, or **10,000** nodes in **60** seconds.

Deploy a HelloWorld App   Getting Started Video   Online Documentation

2 Nodes
18 Commands
11 One Click Launch
3 Recurring Jobs

✓ Open and Cloud Ready
Work on all IaaS cloud
Can Deploy **ANY** Application
100% Open Source

User Friendly
Intuitive Management of Resources
Job Launch Wizard
One Click Launch
Logs with FullText Search

Ready to Use
Everything to manage your application
One click deploy, rollback
Build, Management Task, Monitoring

Scalable and Fast
Built on AKKA and Async HTTP Client
Maximum concurrency
Scale simply and predictably

Simple but Powerful
Setup in 2 minutes, **ZERO** dependency
Simple Command, NodeGroup, and Job
Run generic request on any HTTP based service

About Us
Cronus is a PAAS framework for frictionless application life cycle management in cloud, it is designed and built by Stackscaling, part of the work is based on eBay open source Agent and REST Commander

Home · GitHub
© 2013-2014 stackscaling.com, all rights reserved.

2.2 Command Template

Command template is a complete HTTP(S) request in json format with replaceable variables

Sample synchronous HTTP request

```
{
 "name" : "SAMPLE_SYNC_COMMAND",                                // unique name of the command
 "category" : "sample",                                         // optional category name for grouping
 "httpTaskRequest" : {
   "taskType" : "async",                                        // this is a http request with response
   "httpClientType" : "httpClient",                             // type of http client to use
   "executionOption" : {                                        // below are options for sending reque
     "timeOutSec" : 0,                                          // time out in second
     "initDelaySec" : 0,                                        // initial delay in second
     "maxRetry" : 3,                                            // total # of retry on failure
     "retryDelaySec" : 0                                        // delay before retry
   },
   "urlTemplate" : {                                            // http request template
     "url" : "https://<host>:12020/services/<serviceName>",     // url
     "method" : "POST",                                         // http method
     "body" : {},                                               // http body
     "headers" : {
       "content-type" : "application/json"                      // http headers
     },
     "parameters" : {}                                          // query string parameters
   },
   "resProcessorName" : "agentProcessor"                        // bean for processing http response
 },
 "userData" : {                                                 // user provided data to launch commanc
  "serviceName" : "myservice"
 }
}
```

Sample asynchronous HTTP request with polling for result

```
{
 "name" : "SAMPLE_ASYNCPOLL_COMMAND",
 "category" : "sample",
 "httpTaskRequest" : {
   "taskType" : "asyncpoll",                             // this is a request followed by polling
   "httpClientType" : "httpClient",
   ...                                                   // same url template as above for request
   "monitorOption" : {                                   // options for polling
     "timeOutSec" : 0,
     "initDelaySec" : 0,
     "maxRetry" : 3,
     "retryDelaySec" : 0,
     "intervalSec" : 10                                  // polling interval in second
   },
   "pollTemplate" : {                                    // http polling template
     "url" : "https://<host>:12020/status/<uuid>",
     "method" : "GET",
     "body" : {},
     "headers" : {
       "content-type" : "application/json"
     },
     "parameters" : {}
   },
   "resProcessorName" : "agentPollProcessor",            // bean for processing http response
 },
 "userData" : {
   "serviceName" : "myservice"
 }
}
```

Sample agent request

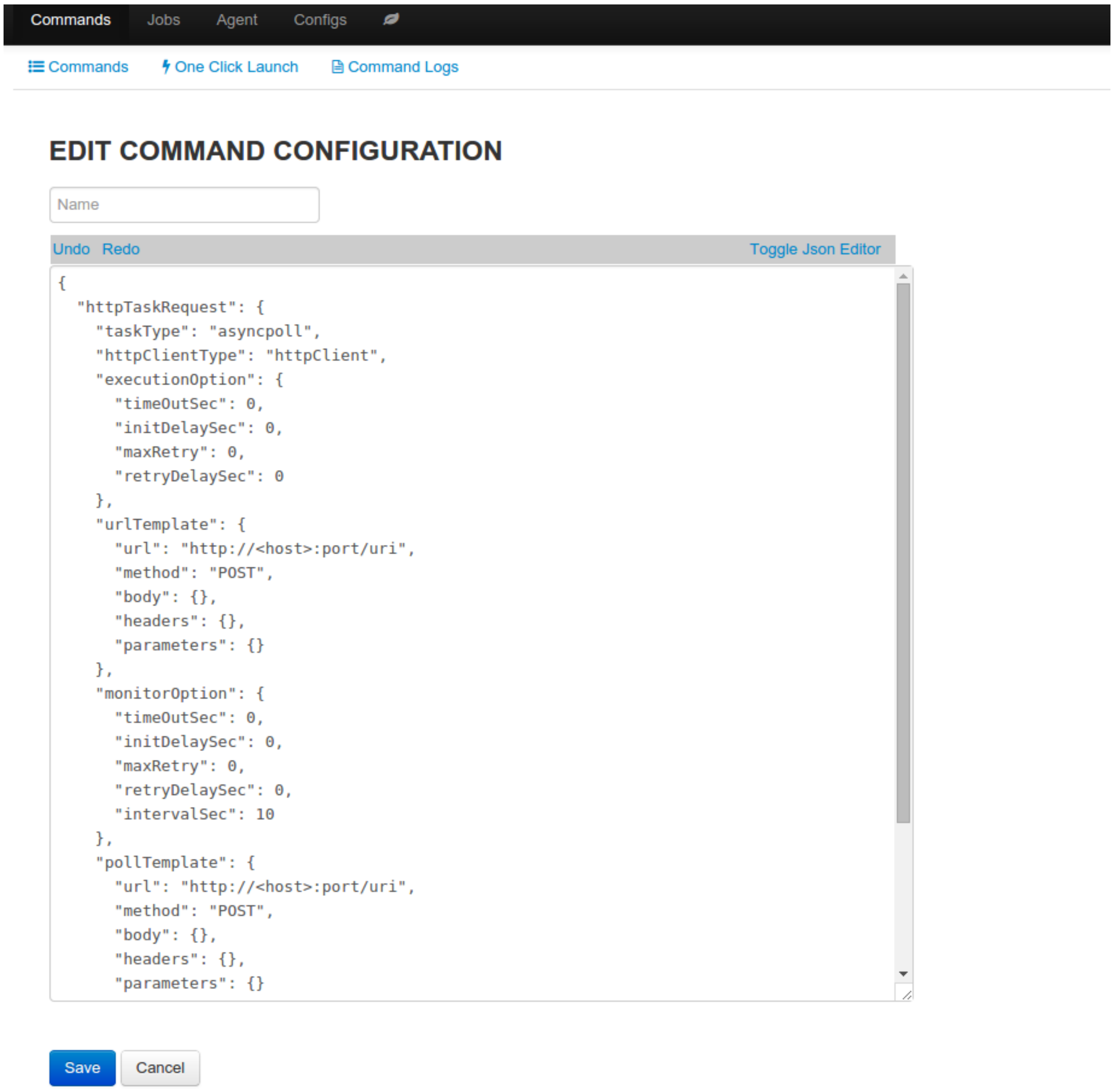
```
{
  "name" : "Agent_Clean_Service",
  "category" : "agent",                 // agent command with many fields predefined and can be skipped
  "httpTaskRequest" : {
    "taskType" : "asyncpoll",
    "httpClientType" : "httpClient",
    "urlTemplate" : {
      "url" : "https://<host>:12020/services/<serviceName>/action/cleanup",
      "method" : "POST",
      "body" : { },
      "headers" : { },
      "parameters" : { }
    }
  },
  "userData" : {
    "serviceName" : "myservice"
  }
}
```

2.3 Command

Create command

1. Command can be created in two ways
   - Create from scratch
   - Clone from an existing command
2. Define command type
   - async: request with reponse as the result
   - asyncpoll: request followed by polling to get result
3. Fill in the command template with http request values, similar to defining http request in RESTful client like POSTMAN
4. Define template parameter; template parameter can be used in all http reuqest values, template parameter is identified by "<>", minimally if the command can be executed against different hosts, "<host>" needs to be part of the url value. Template parameters will be replace with real values at execution time, or if values are not provided, the parameters will be skipped.
5. Define user data, it will be used to drive command launch wizard

edit command



EDIT COMMAND CONFIGURATION

```
{
  "httpTaskRequest": {
    "taskType": "asyncpoll",
    "httpClientType": "httpClient",
    "executionOption": {
      "timeOutSec": 0,
      "initDelaySec": 0,
      "maxRetry": 0,
      "retryDelaySec": 0
    },
    "urlTemplate": {
      "url": "http://<host>:port/uri",
      "method": "POST",
      "body": {},
      "headers": {},
      "parameters": {}
    },
    "monitorOption": {
      "timeOutSec": 0,
      "initDelaySec": 0,
      "maxRetry": 0,
      "retryDelaySec": 0,
      "intervalSec": 10
    },
    "pollTemplate": {
      "url": "http://<host>:port/uri",
      "method": "POST",
      "body": {},
      "headers": {},
      "parameters": {}
```

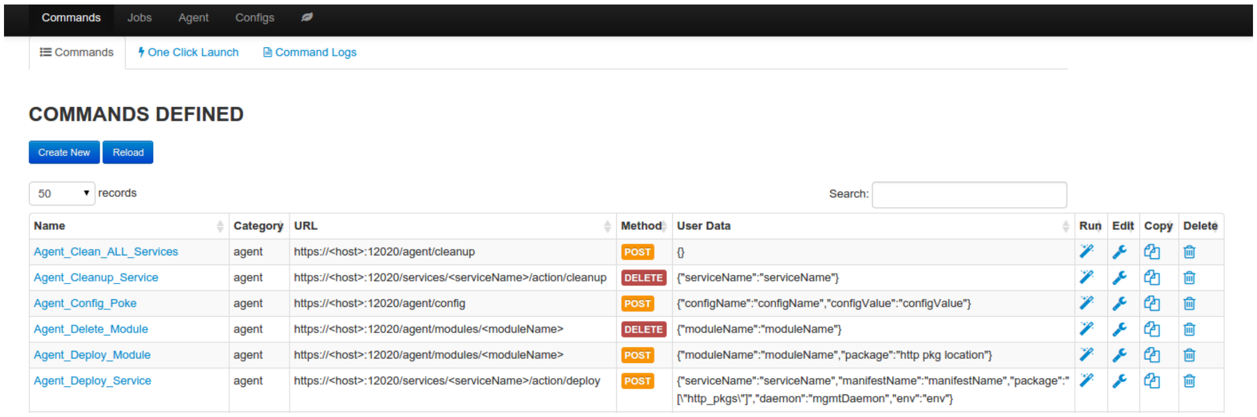
Modify command

Except the name, which is immutable, data of the command can be modified

Delete command

Delete command from persistent store

command summary page



COMMANDS DEFINED

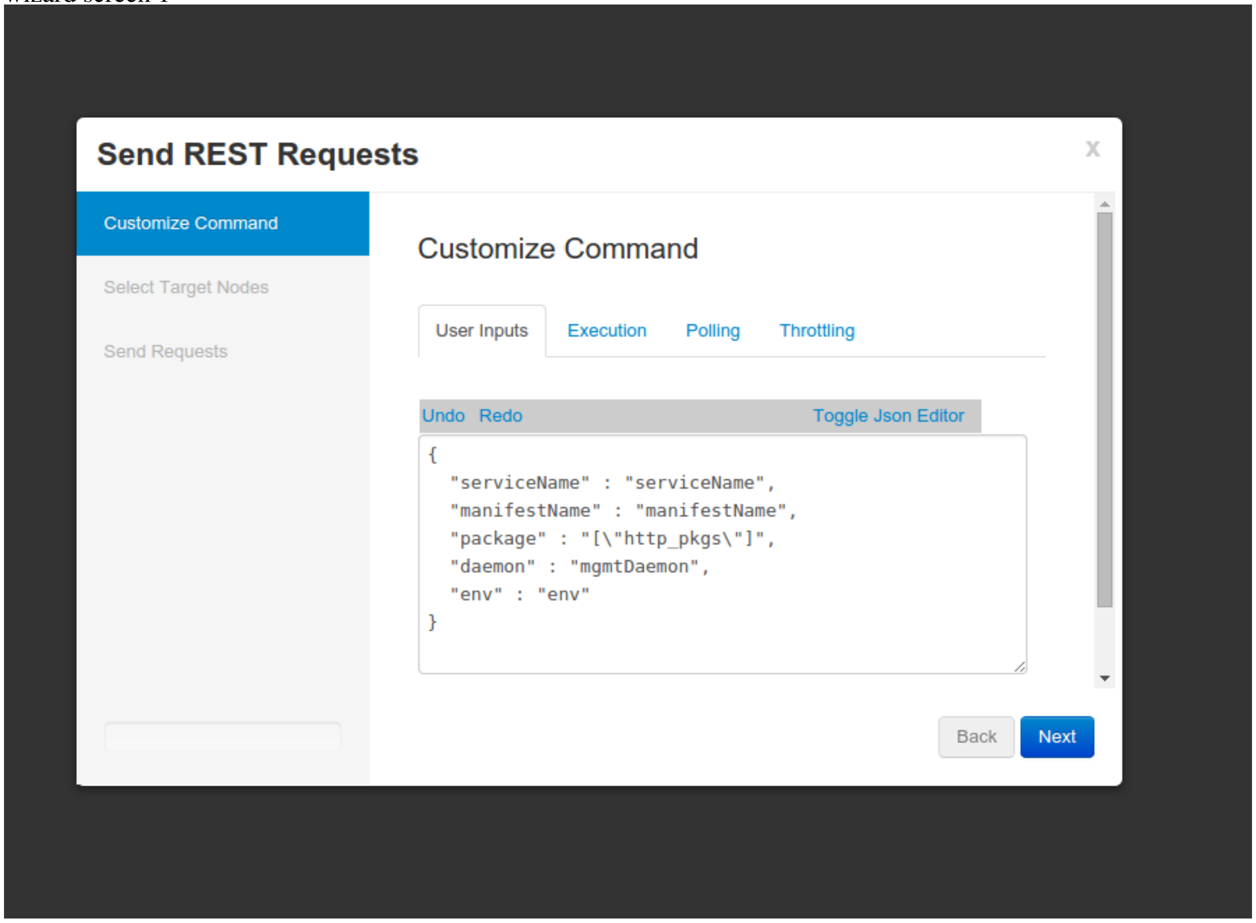
Name	Category	URL	Method	User Data	Run	Edit	Copy	Delete
Agent_Clean_ALL_Services	agent	https://<host>:12020/agent/cleanup	POST	{}				
Agent_Cleanup_Service	agent	https://<host>:12020/services/<serviceName>/action/cleanup	DELETE	{"serviceName":"serviceName"}				
Agent_Config_Poke	agent	https://<host>:12020/agent/config	POST	{"configName":"configName","configValue":"configValue"}				
Agent_Delete_Module	agent	https://<host>:12020/agent/modules/<moduleName>	DELETE	{"moduleName":"moduleName"}				
Agent_Deploy_Module	agent	https://<host>:12020/agent/modules/<moduleName>	POST	{"moduleName":"moduleName","package":"http pkg location"}				
Agent_Deploy_Service	agent	https://<host>:12020/services/<serviceName>/action/deploy	POST	{"serviceName":"serviceName","manifestName":"manifestName","package":"[\"http_pkgs\"]","daemon":"mgmtDaemon","env":"env"}				

Run command

Run command will launch command wizard

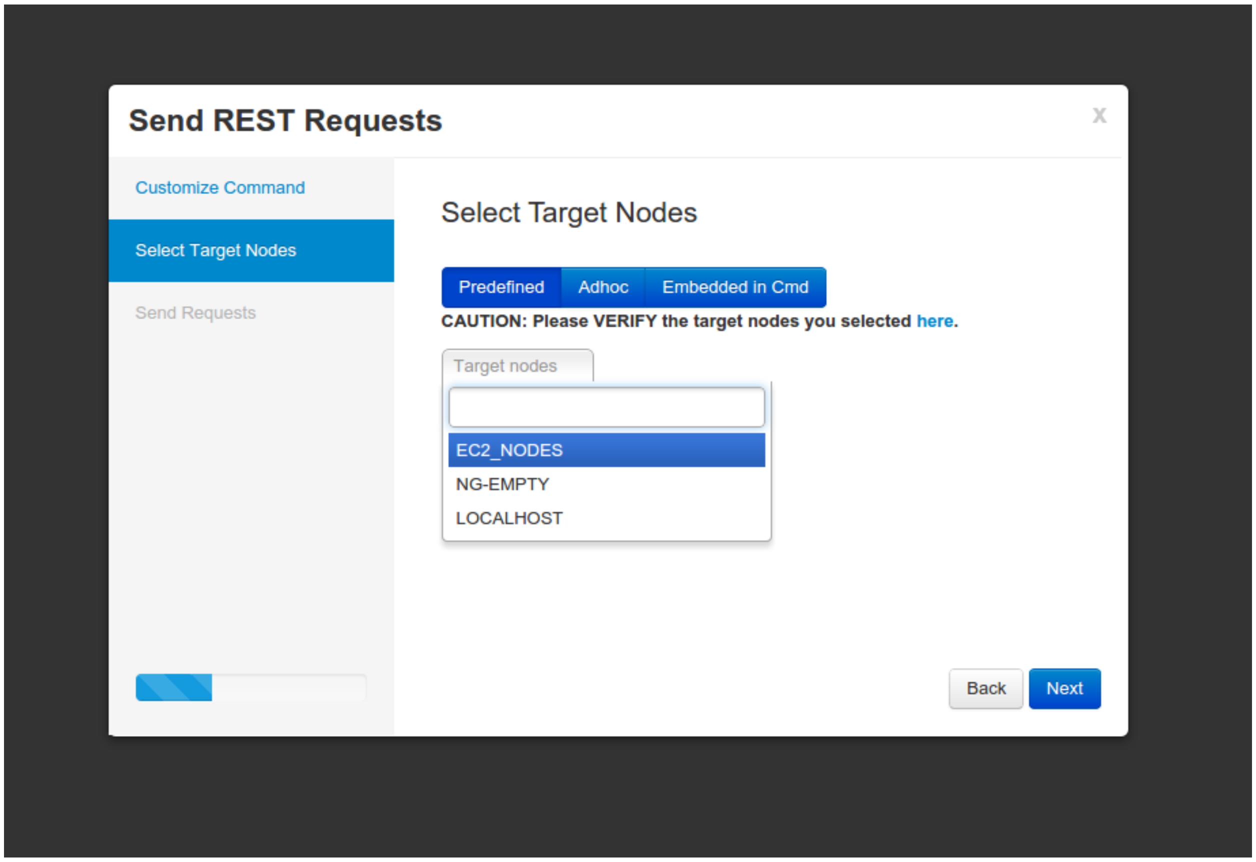
- wizard screen 1: customize command with execution options, user data
- wizard screen 2: select nodegroup against which command will be run
- wizard screen 3: confirm and launch command job, job progress and result can be found in command job log.

wizard screen 1



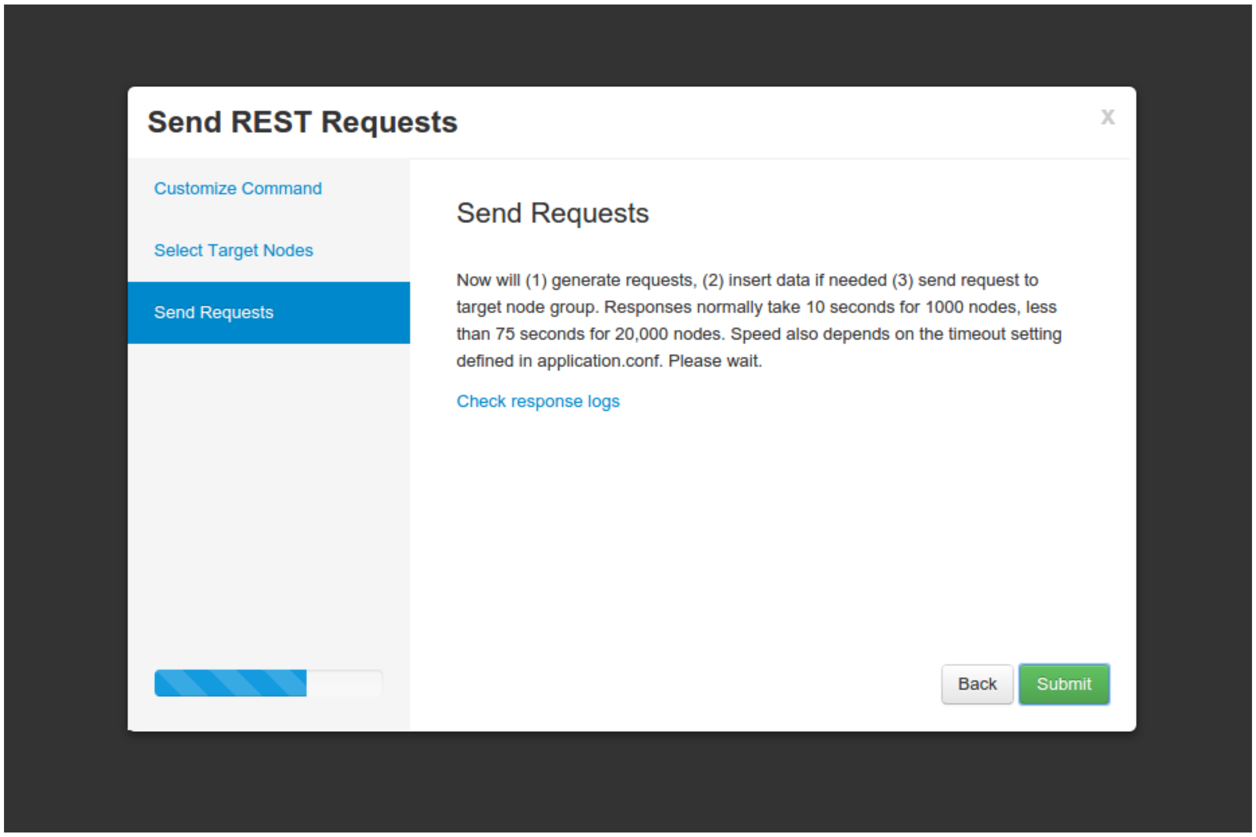
```
{
  "serviceName" : "serviceName",
  "manifestName" : "manifestName",
  "package" : "[\"http_pkgs\"]",
  "daemon" : "mgmtDaemon",
  "env" : "env"
}
```

wizard screen 2



CAUTION: Please VERIFY the target nodes you selected here.

wizard screen 3


Send REST Requests

Send Requests

Now will (1) generate requests, (2) insert data if needed (3) send request to target node group. Responses normally take 10 seconds for 1000 nodes, less than 75 seconds for 20,000 nodes. Speed also depends on the timeout setting defined in application.conf. Please wait.

2.4 OneClick Launch

Oneclick launch captures all the input necessary to launch a command job, this allows a job to be launched with "one click" (or one API call).

Why it is useful

- Deploy or rollback applications with one click or one API call
- Save a job for repeated execution later
- Enhance automation with adaptive smart template parameters, for example can deploy latest application package to target nodes as the application package gets uploaded

Sample oneclick launch

```
{
 "name": "deploy_pyserver_local",                       // unique name of the oneclick command
 "userData": {                                          // custom user data
   "exe_retry": "3",                                    // execution options
   "mon_retry": "3",
   "thrStrategy": "UNLIMITED",
   "mon_int": "10"
   // user data
   "var_values": "{\"serviceName\":\"pyserver\",\"package\":[\"http://<cmInternalIp>:9000/agent/downl
 },
 "commandKey": "Agent_Deploy_Service",                  // command name
 "nodeGroupKey": "LOCALHOST"                            // nodegroup name
}
```

Create oneclick launch

Oneclick launch can be created in two ways

1. From command log tab, save an already executed command job as oneclick launch
2. From oneclick launch tab, create by cloning an existing oneclick launch

Smart template parameters

The following smart template parameters can be used in oneclick launch to allow greater degree of reusability

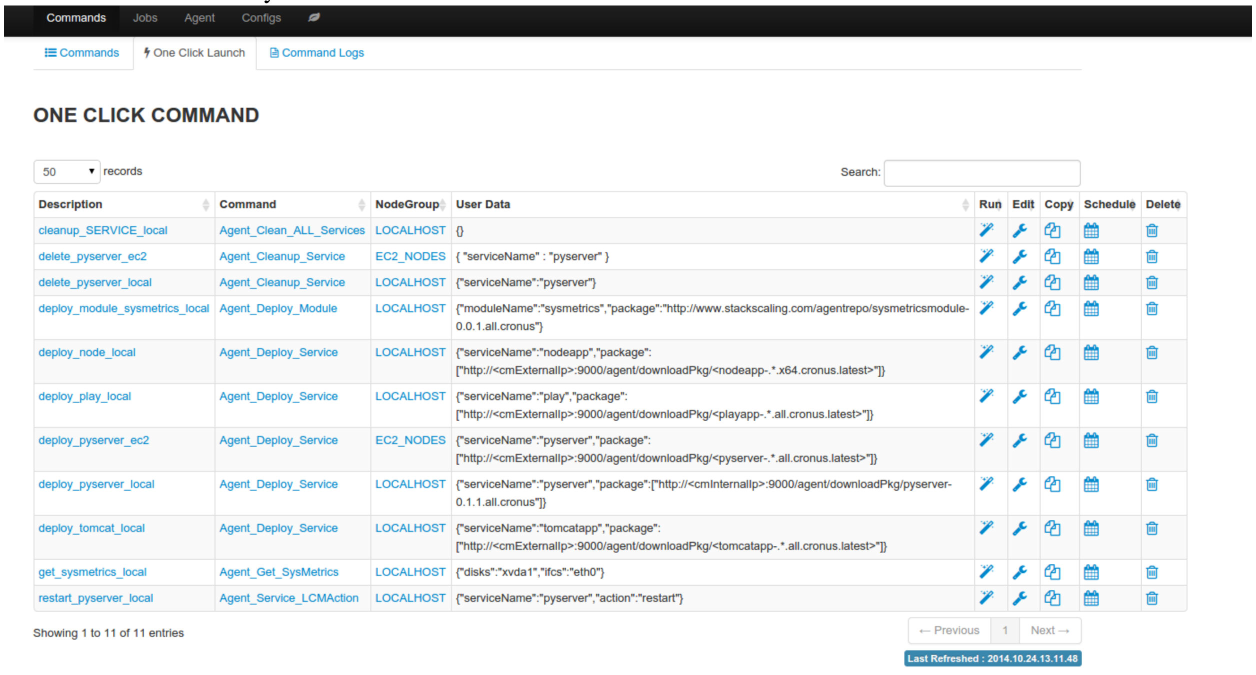
- <cmExternalIp>: reference public IP of the cronus master, value defined in cronusmaster config file
- <cmInternalIp>: reference private IP of the cronus master, value defined in cronusmaster config file
- <service-.*.platform.cronus.latest>: reference latest uploaded package for a service, for example <pyserver-.*.all.cronus.latest> is the latest version of the pyserver package, <pyserver-1.*.all.cronus.latest> is the latest version of 1.x.y of the pyserver package

With smart template parameters, one can create an oneclick launch to deploy latest application package to a group of nodes, without having to change anything in the oneclick launch.

Run oneclick

Run from "oneclick launch" tab with a single click, or one API call, redirect to command job log page upon successful launch.

oneclick launch summary



ONE CLICK COMMAND

50 records Search:

Description	Command	NodeGroup	User Data	Run	Edit	Copy	Schedule	Delete
cleanup_SERVICE_local	Agent_Clean_ALL_Services	LOCALHOST	{}	🪄	🔧	⧉	📅	🗑
delete_pyserver_ec2	Agent_Cleanup_Service	EC2_NODES	{ "serviceName" : "pyserver" }	🪄	🔧	⧉	📅	🗑
delete_pyserver_local	Agent_Cleanup_Service	LOCALHOST	{"serviceName":"pyserver"}	🪄	🔧	⧉	📅	🗑
deploy_module_sysmetrics_local	Agent_Deploy_Module	LOCALHOST	{"moduleName":"sysmetrics","package":"http://www.stackscaling.com/agentrepo/sysmetricsmodule-0.0.1.all.cronus"}	🪄	🔧	⧉	📅	🗑
deploy_node_local	Agent_Deploy_Service	LOCALHOST	{"serviceName":"nodeapp","package": ["http://<cmExternalIp>:9000/agent/downloadPkg/<nodeapp-.*.x64.cronus.latest>"]}	🪄	🔧	⧉	📅	🗑
deploy_play_local	Agent_Deploy_Service	LOCALHOST	{"serviceName":"play","package": ["http://<cmExternalIp>:9000/agent/downloadPkg/<playapp-.*.all.cronus.latest>"]}	🪄	🔧	⧉	📅	🗑
deploy_pyserver_ec2	Agent_Deploy_Service	EC2_NODES	{"serviceName":"pyserver","package": ["http://<cmExternalIp>:9000/agent/downloadPkg/<pyserver-.*.all.cronus.latest>"]}	🪄	🔧	⧉	📅	🗑
deploy_pyserver_local	Agent_Deploy_Service	LOCALHOST	{"serviceName":"pyserver","package":["http://<cmInternalIp>:9000/agent/downloadPkg/pyserver-0.1.1.all.cronus"]}	🪄	🔧	⧉	📅	🗑
deploy_tomcat_local	Agent_Deploy_Service	LOCALHOST	{"serviceName":"tomcatapp","package": ["http://<cmExternalIp>:9000/agent/downloadPkg/<tomcatapp-.*.all.cronus.latest>"]}	🪄	🔧	⧉	📅	🗑
get_sysmetrics_local	Agent_Get_SysMetrics	LOCALHOST	{"disks":"xvda1","ifcs":"eth0"}	🪄	🔧	⧉	📅	🗑
restart_pyserver_local	Agent_Service_LCMAction	LOCALHOST	{"serviceName":"pyserver","action":"restart"}	🪄	🔧	⧉	📅	🗑

Showing 1 to 11 of 11 entries

← Previous 1 Next →

Last Refreshed : 2014.10.24.13.11.48

2.5 Nodegroup

Nodegroup defines a group of nodes with similar characteristics, and similar purpose. For example, a group of nodes that install the same operating system and cronus agent, in the same network, and are running the same application.

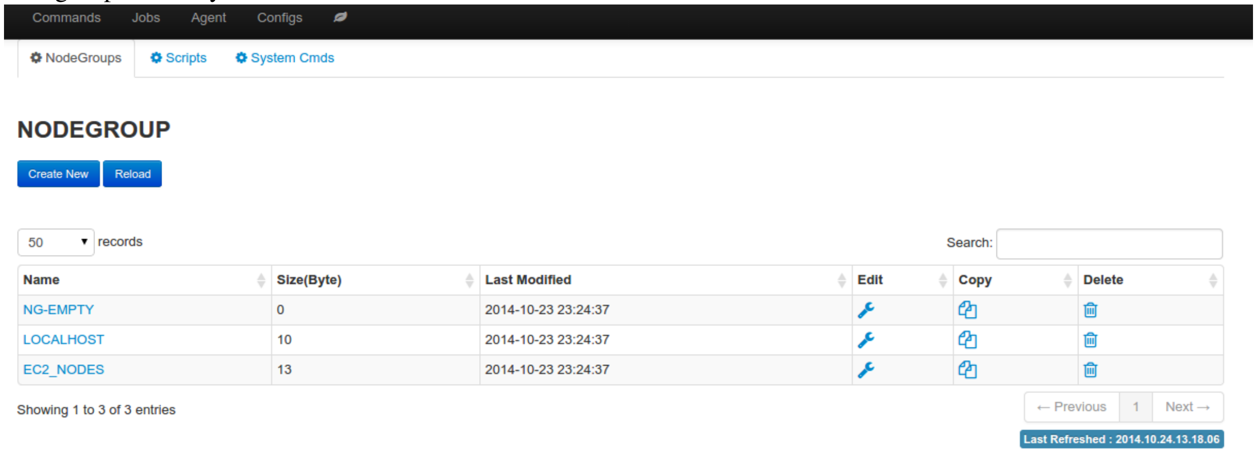
Create nodegroup

Nodegroup is uniquely identified by its name, values are line separated hostnames or IPs.

Use nodegroup

- Can be referenced in command wizard when run command job
- Can be referenced in oneclick launch
- Can be referenced in recurring job

nodegroup summary

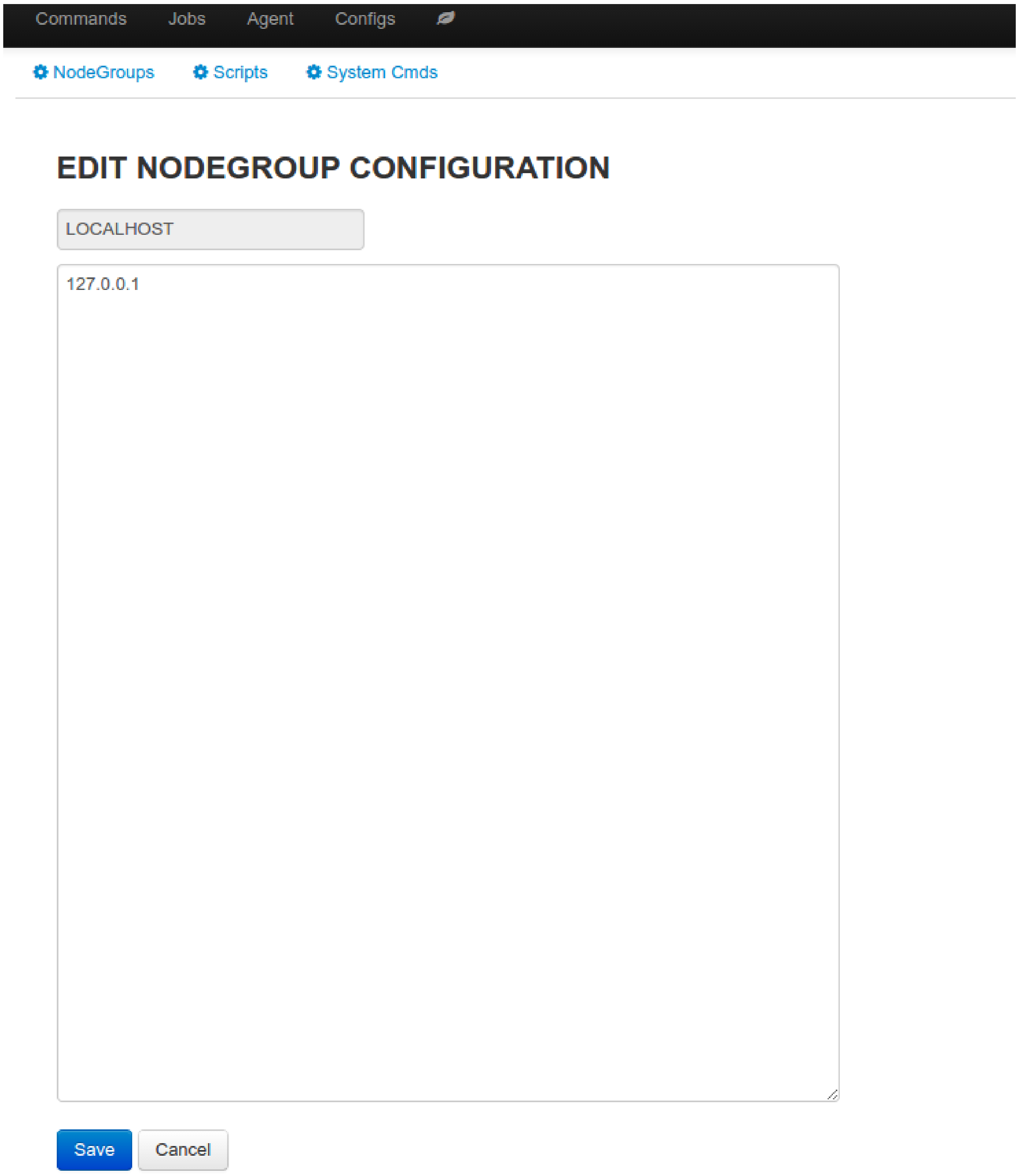


Name	Size(Byte)	Last Modified	Edit	Copy	Delete
NG-EMPTY	0	2014-10-23 23:24:37			
LOCALHOST	10	2014-10-23 23:24:37			
EC2_NODES	13	2014-10-23 23:24:37			

Showing 1 to 3 of 3 entries

Last Refreshed : 2014.10.24.13.18.06

edit nodegroup



EDIT NODEGROUP CONFIGURATION

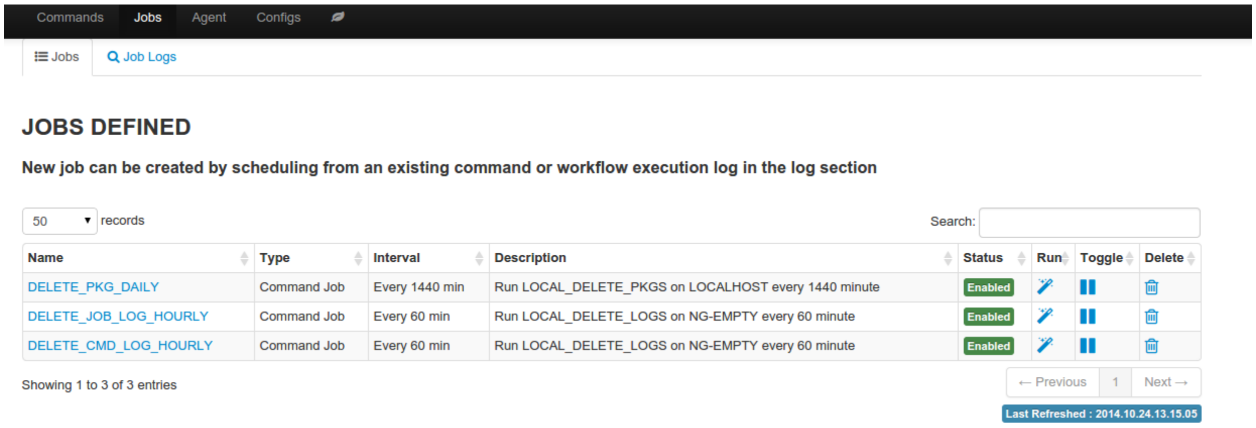
LOCALHOST

127.0.0.1

2.6 Recurring Job

An existing one time command job or oneclick launch can be made to run repeatly as a recurring job.

recurring jobs



JOBS DEFINED

New job can be created by scheduling from an existing command or workflow execution log in the log section

50 records   Search:

Name	Type	Interval	Description	Status	Run	Toggle	Delete
DELETE_PKG_DAILY	Command Job	Every 1440 min	Run LOCAL_DELETE_PKGS on LOCALHOST every 1440 minute	Enabled			
DELETE_JOB_LOG_HOURLY	Command Job	Every 60 min	Run LOCAL_DELETE_LOGS on NG-EMPTY every 60 minute	Enabled			
DELETE_CMD_LOG_HOURLY	Command Job	Every 60 min	Run LOCAL_DELETE_LOGS on NG-EMPTY every 60 minute	Enabled			

Showing 1 to 3 of 3 entries

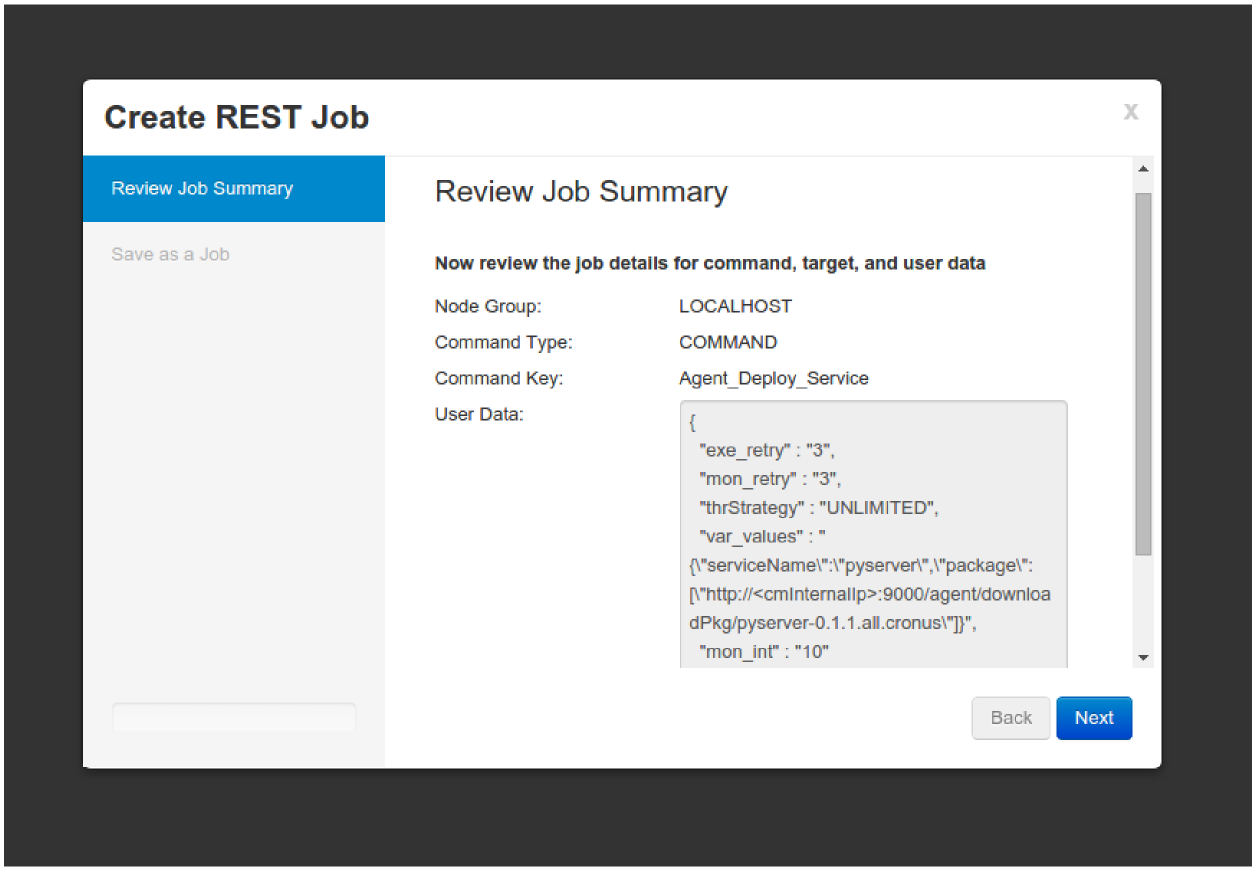
← Previous 1 Next →

Last Refreshed : 2014.10.24.13.15.05

Create Recurring Job

1. Recurring job can be created in two ways
   - Create from an existing command job from command job log
   - Create from an existing oneclick launch
2. Go through the recurring job wizard to define a name uniquely identified the job, and recurring frequency in the interval of 5 minutes
3. Once saved, job running state can be toggled on and off from the Job page
4. Job log of previously launched job can be found in job log

job wizard screen 1



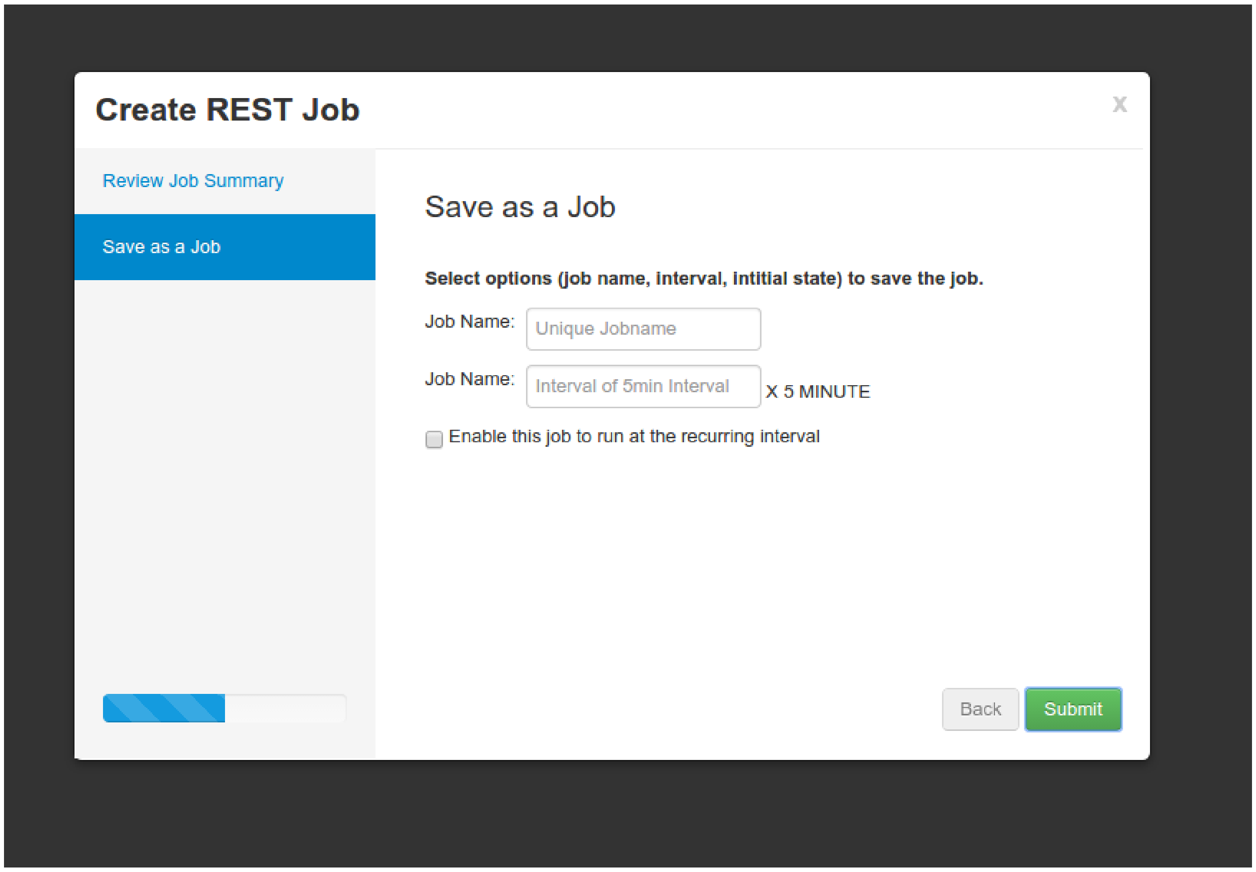
Review Job Summary

Now review the job details for command, target, and user data

Node Group:	LOCALHOST
Command Type:	COMMAND
Command Key:	Agent_Deploy_Service
User Data:	

```
{
  "exe_retry" : "3",
  "mon_retry" : "3",
  "thrStrategy" : "UNLIMITED",
  "var_values" : "
{\"serviceName\":\"pyserver\",\"package\":
[\"http://<cmInternalIp>:9000/agent/downloa
dPkg/pyserver-0.1.1.all.cronus\"]}",
  "mon_int" : "10"
```

job wizard screen 2



Save as a Job

Select options (job name, interval, initial state) to save the job.

Job Name: Unique Jobname

Job Name: Interval of 5min Interval X 5 MINUTE

☐ Enable this job to run at the recurring interval

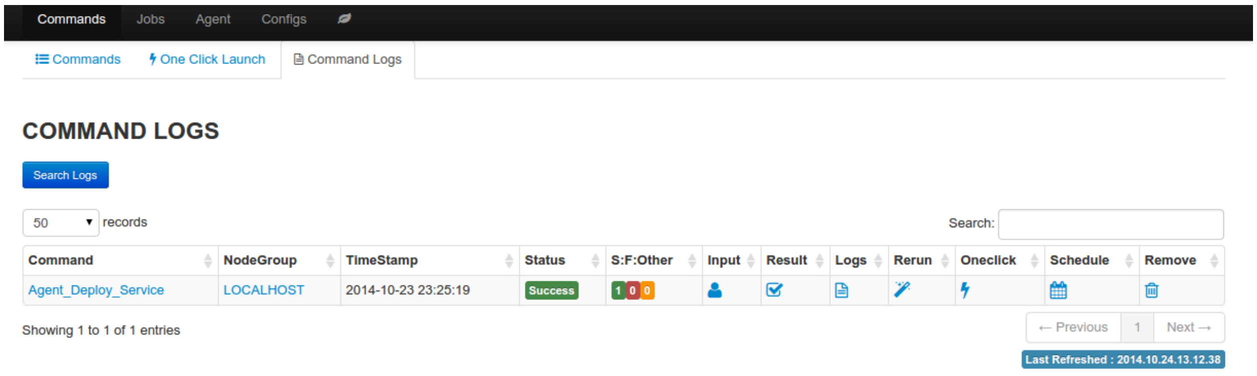
2.7 Command Log

Log provide important information about result of command job, and provide chance to create new oneclick launch and recurring job.

Information available from the log

- status: status of the job (running|success|failure)
- S:F:Other: number of nodes for each status value
- input: customize user data
- result: summary of the result in json format, the raw response form the target node are truncated to 200 characters
- logs: full text search of the job result, as well as raw logs from the target node
- rerun: run the same command job again
- oneclick: create a new oneclick launch from the command job
- schedule: create a recurring job from the command job
- remove: delete the particular log, as well as related full text search entry

command logs

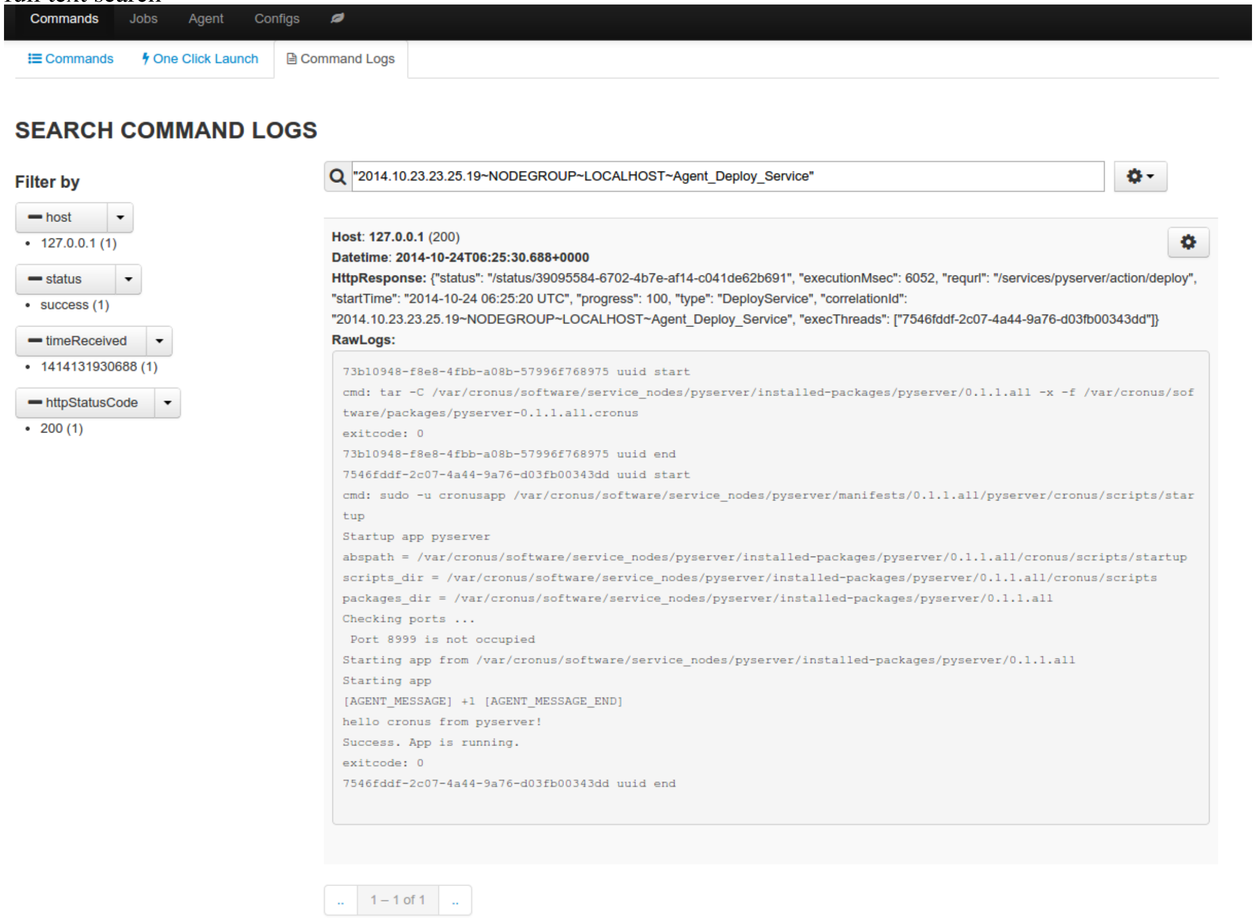


COMMAND LOGS

Command	NodeGroup	TimeStamp	Status	S:F:Other	Input	Result	Logs	Rerun	Oneclick	Schedule	Remove
Agent_Deploy_Service	LOCALHOST	2014-10-23 23:25:19	Success	1 0 0							

Showing 1 to 1 of 1 entries

full text search



SEARCH COMMAND LOGS

Filter by

- host
  - 127.0.0.1 (1)
- status
  - success (1)
- timeReceived
  - 1414131930688 (1)
- httpStatusCode
  - 200 (1)

"2014.10.23.23.25.19~NODEGROUP~LOCALHOST~Agent_Deploy_Service"

Host: **127.0.0.1** (200)
Datetime: **2014-10-24T06:25:30.688+0000**
HttpResponse: {"status": "/status/39095584-6702-4b7e-af14-c041de62b691", "executionMsec": 6052, "requrl": "/services/pyserver/action/deploy", "startTime": "2014-10-24 06:25:20 UTC", "progress": 100, "type": "DeployService", "correlationId": "2014.10.23.23.25.19~NODEGROUP~LOCALHOST~Agent_Deploy_Service", "execThreads": ["7546fddf-2c07-4a44-9a76-d03fb00343dd"]}
RawLogs:

```
73b10948-f8e8-4fbb-a08b-57996f768975 uuid start
cmd: tar -C /var/cronus/software/service_nodes/pyserver/installed-packages/pyserver/0.1.1.all -x -f /var/cronus/sof
tware/packages/pyserver-0.1.1.all.cronus
exitcode: 0
73b10948-f8e8-4fbb-a08b-57996f768975 uuid end
7546fddf-2c07-4a44-9a76-d03fb00343dd uuid start
cmd: sudo -u cronusapp /var/cronus/software/service_nodes/pyserver/manifests/0.1.1.all/pyserver/cronus/scripts/star
tup
Startup app pyserver
abspath = /var/cronus/software/service_nodes/pyserver/installed-packages/pyserver/0.1.1.all/cronus/scripts/startup
scripts_dir = /var/cronus/software/service_nodes/pyserver/installed-packages/pyserver/0.1.1.all/cronus/scripts
packages_dir = /var/cronus/software/service_nodes/pyserver/installed-packages/pyserver/0.1.1.all
Checking ports ...
 Port 8999 is not occupied
Starting app from /var/cronus/software/service_nodes/pyserver/installed-packages/pyserver/0.1.1.all
Starting app
[AGENT_MESSAGE] +1 [AGENT_MESSAGE_END]
hello cronus from pyserver!
Success. App is running.
exitcode: 0
7546fddf-2c07-4a44-9a76-d03fb00343dd uuid end
```

2.8 Configurations

This section explains cronus master configurations

Basic configurations

Cronus master configuration section in a sample play application.conf

```
# use local file | aws_s3 | openstack_swift to store user data
agentmaster.userDataDao=file
agentmaster.logProgressEnabled=true

# cronusmaster public and private IP if any
agentmaster.externalIp=some_external_ip
agentmaster.internalIp=some_internal_ip

# user data location (applicable only if use local fs for user data)
#agentmaster.userDataDao.file.dir=.appdata

# s3 related configs (applicable only if use aws_s3 for user data)
agentmaster.userDataDao.s3.myAccessKeyID=your_access_key_id
agentmaster.userDataDao.s3.mySecretKey=your_secret_key

# swift related configs (applicable only if use swift for user data)
agentmaster.userDataDao.swift.tenantId=your_tenant_id
agentmaster.userDataDao.swift.tenantName=your_tenant_name
agentmaster.userDataDao.swift.username=your_username
agentmaster.userDataDao.swift.password=your_password
agentmaster.userDataDao.swift.authenticationUrl=authn_url

# elastic search
# use embedded or remote elastic search backend
agentmaster.localEsEnabled=true
# location of search index (applicable only if use embedded elastic search)
agentmaster.esData=user_data/elasticsearch_data/data
# search end point (applicable only if use remote elastic search)
agentmaster.esEp=some_search_host

# agent basic authentication
agentmaster.cronusagent.password=username:password
# agent PKI authentication
agentmaster.cronusagent.pkicert=path_to_cert

# ... rest of the play configuration in application.conf
```

Configuration override for different environment

One can build separate application.conf.{environment} in the play conf/ directory to be used for deploying cronus master in different environment. Upon deployment time

1. Pass the environment value through agent deploy API call, e.g. `curl -X POST '{..., "env": "prod"}' https://host:12020/services/cronusmaster/action/deploy`
2. Agent will try to find the configuration file with .prod postfix and use it (by renaming it to application.conf)
3. Or if the environment specific configuration file is not found, will use the default base configuration application.conf

Same schemes work for logging configurations (log4j.properties) as well.

2.9 CronusMaster APIs

A few CronusMaster APIs can be called directly with json response.

- GET /commands/oneclickJson : list all oneclick launch

- GET /commands/oneclickRunJson?dataId={oneclick_launch_name} : execute oneclick launch

Additional APIs that drives CronusMaster UI can be found in play “routes” file.

CHAPTER 3

Cronus Agent

3.1 Install in IaaS Cloud

Follow the steps to install agent in AWS EC2, or Goggle GCE Cloud

Prerequisites

- linux (Ubuntu, Cent, Redhat, Fedora etc.)
- sudo permission
- python > 2.6.7 && < 3
- openssl > 0.9.8
- wget
- system management daemon systemd or upstart

Install in dev (default ssl cert, no auth)

```
# install agent
wget -qO- 'http://cronuspaas.github.io/downloads/install_agent' | sudo dev=true bash
# check agent availability
curl -k https://localhost:12020/agent
```

Install in prod

```
# install agent
wget -qO- 'http://cronuspaas.github.io/downloads/install_agent' | \
sudo server_pem=path_to_ssl_cert agent_pwd=user:password bash
# check agent availability
curl -k https://localhost:12020/agent
```

3.2 Application Lifecycle

This section describes application lifecycle defined by Cronus framework.

3.2.1 Terminology

- Service: an application, one or more instances serving the same function and code.

- Manifest: a version of an application, composed of one or more packages. Multiple manifests may be deployed to a ServiceInstance, but only one can be active at a time.
- Active Manifest: a manifest actively in use.
- Package: the smallest unit of deployment that Cronus understands. It can be a tarball containing application runtime, code, config, and cronus structure required for deployment.
- Cronus Package: a cronus structure that is zip-tarred as a {service}-{version}.{platform}.cronus package.
- ServiceInstance - is a single instance of a service, typically mapped to a single compute.

3.2.2 Application Hierarchy

- **A service have one or many manifests, among which, 0 or 1 active manifest, one manifest can have one or many cronus pa**
  service (1) –> (1..n) manifests (1) –> (1..n) packages service (1) –> (0..1) active_manifest_x
- **A service deployed in cloud have one or many compute, each deployed with the active manifest of the same version**
  service_deployed_in_cloud –> (1..n) service_instances (1) –> active_manifest_x

3.2.3 Application Life Cycle

- Service lifecycle: Service can be created on a compute, and later can be deleted if no active manifest presents.
- Manifest lifecycle: Manifest can be created on a compute for a particular service, activate a manifest will deactivating any existing active manifest and make the manifest active manifest, active manifest can be startup, shutdown, and restart, active manifest can be deactivated and then deleted.

3.2.4 Application Package Structure (aka Cronus Package)

- **Cronus Package** All packages within an application manifest need to be packaged with certain structure and provide a set of life cycle scripts

```
package_root/              # package root directory
    cronus/                # cronus directory
        cronus.prop        # prop file describe the package
        cronus.ini         # configuration file in json format to customize agent behavior
        scripts/           # optional scripts directory containing package scripts
            install        # execute when package is untarred the first time, executed once
            activate       # execute when parent manifest has been set active.
            startup        # execute to 'start' a package
            running        # execute to check package runtime state
            shutdown       # execute to 'shut down' a package
            deactivate     # execute when parent manifest gets deactivated
    ...                    # any application files or directories to be packaged
```

- Cronus package deployed on compute

```
/path_to_service_home/{service}      # service root directory
   .metadata.json                    # metadata json file
   .appdata/                         # application data directory, shared among manifests
   manifests/                        # application manifests
      active                         # symlink point to active manifest
      1.0.0/                         # manifest 1.0.0
         pkg1                        # inflated cronus package1
```

- **Naming Convention** Tarred cronus package has to follow the naming convention of "{packageName}-{majorVersion}.{minorVersion}.{incrementalVersion}.[optionalVersionSuffix].{platform}.cronus", e.g. agent-1.0.0.x86_ubuntu.cronus An optional corresponding property file that specify metadata of the cronus package, most importantly md5 checksum of the cronus package, property file has to follow the naming convention of "{sameCronusPackageName}.prop", e.g. PortalWebAppConfiguration-1.0.0.unix.cronus.prop, for example

3.2.5 Application Life Cycle Management

Life Cycle Management Scripts

Agent requires a set of LCM scripts in order to control the application life cycle precisely

Script	Required	Description
install	optional	additional installation operations after software package is uncompressed and manifest created run only once within manifest life time
activate	optional	activate manifest run once every time manifest is activated, or reset
startup	required	start the application run once every time application is activated, startup, restart, or reset
shutdown	optional	shutdown the application run once every time application is activated, shutdown, restart, or reset
deactivate	optional	deactivate the application run once every time manifest is activated, or reset

- Because application startup script is called by a process launched by agent, one must make sure that
- Startup script returns without blocking
- Fork the application process to a separate process
- Detach the application process from its parent process, use setsid() to make the new process new group leader so that it does not terminate when agent process shutdown/restart. For more details see http://stackoverflow.com/questions/2613104/why-fork-before-setsid

Passing Parameters

- Default environment variables: agent injects the following environment variables to application before invoking application life cycle scripts
  - $CRONUSAPP_HOME: absolute path to the application service root directory
  - $LCM_CORRELATIONID: correctionid if any passed for the LCM API
- Additional environment variables: any additional parameters passed in through agent deploy API are made available to LCM scripts as environment variables

Timeouts and Passing Information to Agent

- Timeout: Scripts must exit before timeout expires, or process be killed, default timeout is 15 minutes configurable by agent config.
- Progress Timeout: Scripts must demonstrate progress (progress number increasing) by passing progress information to agent or process be killed, default progress timeout is 15 minutes configurable by agent config.
- Passing Information to Agent while Running: Scripts can pass progress and other information to agent via stdout while running, in syntax

```
[AGENT_MESSAGE]
{
    "progress": 50,
}
[AGENT_MESSAGE_END]
```

- Passing Result to Agent at Exit: Scripts can pass result to agent via its return code and stdout.
  - 0: success
  - Any non-zero return code: failure
  - Additional information including status, progress, and message can be passed to agent via stdout, in syntax

    ```
    // sample message to agent for progress or for success
    [AGENT_MESSAGE]
    {
      "progress": 100,
      "result":[
        {"key": {result_key}, "value": {result_value}}
      ]
    }
    [AGENT_MESSAGE_END]

    // sample message to agent for error

    [AGENT_MESSAGE]
    {
      "error": {error_code},
      "errorMsg": {error_message},
      "result":[
        {"key": {result_key}, "value": {result_value}}
      ]
    }
    [AGENT_MESSAGE_END]
    ```

  - Stdout or Stderr: while executing application script, agent reads from stdout and process any message matches the above format and use it to update status. If script fails with non-zero return code, agent reads from stderr, or if it is missing, from last readout from stdout, for anything matches the above format and update error status. Both status can be retrieved from agent status API "/status/:uuid"
  - Mutli-line support: with single-line output, [AGENT_MESSAGE_END] can be omitted. With mutli-line output, agent looks for [AGENT_MESSAGE_END] as end of message indicator, there is a limit of 8k for agent message
  - Encoding: only ascii is supported, other encoding will be skipped

3.3 Application Lifecycle Scripts

This guide describe how to create application lifecycle scripts.

1. Bootstrap your application with cronus package structure and lifecycle script skeleton

   ```
   cd application_root
   wget -qO- 'http://cronuspaas.github.io/downloads/bootstrap_cronus' | DIR=. bash
   ```

2. Modify scripts to your need

   - **Startup script have nodaemon mode, distinguished by cmd line parameter $1="nodaemon"**

    - in nodaemon mode, application process is managed by external service manager (systemd or upstart), in nodaemon mode startup script needs to start the application and block until application exit.
    - in daemon mode, use exec or nohup to fork your application main process, also use setsid to detach your application process from parent agent process.
  - All parameters passed in from agent deploy API call are made available to lifecycle scripts as environment parameters except startup and shutdown scripts, reason being the two script can be executed multiple times out of context of a deployment.
  - Locally persisted application data that have lifecycle beyond the manifest can be accessed through .appdata symlink under the package root directory.

3. Test locally make sure scripts are working as expected
   - Optionally delete the scripts that are not needed
4. Package and upload, test deploy remotely in the cloud

This recipe describes steps to package an application ready for cronus deployment

- Bootstrap application with packaging script

` cd app_root wget -qO- 'http://cronuspaas.github.io/downloads/bootstrap_agent' | bash ` * Modify package.sh with appropriate values for appname, version, src_pkgs, and platform * Package the application: ./package.sh * Package and upload to cronusmaster ` export cronusmaster=_cronusmaster_ip_ ./package.sh upload `

Expected Outcome

- Cronus package and its prop file in app_root/target_cronus/appname-version.platform.cronus
- If uploaded, check package available in cronusmaster UI under Agent->Package

3.4 Deploy Application

This guide describes how to deploy an application directly through agent REST API without cronusmaster

Requires:

- Cronus packaged application HTTP accessible by agent on target machine

How:

```
curl -k -X POST -d '{"package": ["{cronus_package_url}"], "manifest": "{manifest_name}", \
"env": "{env}", "daemon": "upstart|systemd"}' \
https://host:12020/services/{service_name}/action/deploy

# service_name: mandatory service name
# cronus_package_url: mandatory url location to your cronus package
# manifest_name: optional manifest name, default to use cronus package version
# env: optional env value, used to select environment specific application configurations
# daemon: optional daemon manager, used to launch applications managed by upstart or systemd
```

Expected Outcome:

- Service is created if not already exist
- Manifest is created if not already exist
- Any existing active manifest is shutdown, deactivate

- New manifest is install, activate, and startup

3.5 Application Data

This guide describes how to access locally persisted application data

Data local to specific manifest

Any data within the manifest package, or generated after activation in the manifest package is local to the specific manifest only. It can be deleted by agent after the manifest is deactivated and garbage collected

Data local to specific service

Agent creates .appdata under service root directory and creates symlink .appdata in each cronus package root during manifest activation. Data persisted in .appdata and its subdirectories is accessible across all manifests of the service. It can be deleted by agent when the service is deleted.

```
- service_root
    - .appdata
    - manifests
        - active
            - package1
                - .appdata (symlink to service_root/.appdata)
```

Environment variables

Any data passed in from deploy API will be available to the application life cycle script as environment variables

```
# deploy API call
    curl -k -X POST -d '{"package": ["http://$_cronus_package_url_"], "env": "production"' \
    https://host:12020/services/$_service_name_/action/deploy

# in activate script, this will print env=production
    echo "env=$env"
```

3.6 Basic Authentication

This guide describe

- How to install agent with SSL cert
- How to enable agent basic authentication

```
wget -qO- 'http://cronuspaas.github.io/downloads/install_agent' \
| sudo server_pem={path_to_pem_file} agent_pwd={user:password} bash
```

Expect outcome

- Agent up and running with custom SSL cert
- Agent CUD APIs requires http Basic authentication (HTTP header "Authorization: Basic {base64_encoded_user:password}")

3.7 Agent Upgrade

This section describes how to upgrade an existing Agent

New agent package must exist in release folder in git http://github.com/cronuspaas/pkgs/{target_version}.unix.cronus

```
curl -k -H "content-type:application/json" -X POST -d '{"version": "version_eg_0.1.46"}' \
https://host:12020/agent/selfupdate
```

or

Update through cronusmaster UI

Expect Outcome:

Agent upgraded to new version, version number can be found in agent VI page https://host:12020/agent/ValidateInternals.html

3.8 Security Best Practice

Why:

- Agent has elevated access to resource required by its workload
- Agent is mission critical service, interference is not desirable
- Agents are deployed everywhere in target infrastructure

Existing Agent Security Mechanisms:

- All traffic through agent API on HTTPS
- Basic HTTP authentication or PKI based authentication
- Separation of agent user and app user, application managed by agent does not have same privilege as agent has
- Agent and app users are setup as system accounts with login disabled

Security Best Practice:

- Do not enable access to agent API from internet without any restriction. It's best to only allow access from intranet via cronusmaster, or if that's not feasible, only allow access from internet from trusted IPs
- Do NOT use default SSL certs when install agent, always provide your own certs
- Do NOT leave agent API open without authentication, at least set basic auth with your own user:password
- Do NOT keep your credentials in your source code, or encrypt it with your private key

3.9 Agent API List

- "GET /services/" : list all services
- "GET /services" : list all services
- "POST /services/{service}" : create service
- "DELETE /services/{service}" : delete service
- "GET /services/{service}" : list manifests of service
- "GET /services/{service}/action/log" : view service log
- "POST /services/{service}/action/updatemetadata" : set service metadata
- "GET /services/{service}/action/getmetadata" : get service metadata as json

- "GET /services/{service}/action/getmetadataprops" : get service metadata as property format
- "GET /services/{service}/action/listpackages" : list all packages of the service
- "POST /services/{service}/manifests/{manifest}" : create manifest
- "DELETE /services/{service}/manifests/{manifest}" : delete manifest
- "GET /services/{service}/manifests/{manifest}" : list all manifests
- "GET /services/{service}/manifests/{manifest}/action/log" : view manifest log
- "POST /services/{service}/manifests/{manifest}/action/activate" : activate manifest
- "POST /services/dist/startdownload" : download cronus package
- "POST /services/dist/validatepackage" : validate md5sum of cronus package against its prop file
- "GET /services/dist/listpackages" : list all packages
- "POST /services/dist/uploadpackage" : upload a cronus package
- "DELETE /services/dist/{package}" : delete a package
- "GET /status/{uuid}" : get status of a job (from job uuid)
- "DELETE /status/{uuid}" : cancel a job (from job uuid)
- "GET /status/uuidoutput/{uuid}" : get raw logs of a job (from job uuid)
- "GET /status/guidoutput/{guid}" : get raw logs of a job (from pass in guid)
- "GET /agent/logs" : view agent logs
- "GET /agent/logs/{fileName}" : view one agent log file
- "GET /agent" : view agent ValidateInternals page
- "GET /agent/ValidateInternals" : agent ValidateInternals in json format
- "GET /agent/ValidateInternals.html" : agent ValidateInternals html page
- "GET /agent/agenthealth" : agent health information in json format
- "GET /agent/monitors" : agent run monitoring value snapshot
- "GET /agent/threads" : agent thread information in json format
- "POST /services/{service}/action/restart" : restart service
- "POST /services/{service}/action/reset" : reset current active manifest (shutdown, deactivate, activate, startup)
- "POST /services/{service}/action/startup" : startup service
- "POST /services/{service}/action/shutdown" : shutdown service
- "POST /services/{service}/useraction/{useraction}" : run user provide script
- "POST /services/{service}/action/deactivatemanifest" : deactivate current active manifest
- "POST /services/{service}/action/deploy" : deploy a service, create service, manifest and activate
- "POST /services/{service}/action/rollback" : rollback a service to last activated version
- "DELETE /services/{service}/action/cleanup" : delete everything of a service
- "POST /agent/cleanup" : delete all services except agent from the host
- "POST /agent/shutdown" : shutdown agent
- "POST /agent/safeshutdown" : shutdown agent when agent is idle (no activity)

- "POST /agent/selfupdate" : upgrade agent
- "DELETE /agent/selfupdate" : cancel an agent upgrade
- "GET /agent/config" : get agent config
- "POST /agent/config" : override agent config
- "DELETE /agent/config" : delete an agent config override
- "POST /agent/cleanproc" : cleanup all cronusapp processes
- "GET /agent/api" : list this agent API
- "POST /agent/gc" : ondemand agent GC
- "GET /agent/servicesinfo" : list all services and everything of the services
- "GET /agent/modules" : get list of agent modules
- "GET /agent/modules/{module}" : get information of a agent module
- "POST /agent/modules/{module}" : create an agent module
- "DELETE /agent/modules/{module}" : delete an agent module
- "POST /admin/executeScript" : execute script
- "POST /admin/executeCmd" : execute a shell command
- "GET /security/listkeys" : list PKI keys
- "POST /security/key/{key}" : create PKI key
- "DELETE /security/key/{key}" : delete PKI key
- "POST security/validatetoken" : validate an existing token

3.10 Agent Configuration

Agent configurations and default values, configurations can be override through API call

Agent Configurations

```
agent_health_report_interval=120                # interval when agent health is checked
agent_thread_timeout=600                        # agent thread timeout
app_restart_init_delay=15                       # artificial delay before agent starts services after
download_skip_prop=true                         # not require cronus prop file
download_thread_attempt=1                       # download retry
download_thread_max_time=7200                   # download timeout ceiling
download_thread_min_progress_time=10            # download progress timeout
download_thread_min_time=60                     # download timeout floor
download_thread_rate_per_sec=102400             # download rate throttle
exec_thread_progress_timeout=60                 # task execution thread progress timeout
exec_thread_sleep_time=1                        # interval when check task execution progress
exec_thread_timeout=600                         # task execution thread timeout
health_agent_file_descriptor_usage_threshold=500 # max number of file descriptor
health_agent_mem_usage_threshold=200000         # max memory
health_disk_usage_gc_buffer=20                  # agent disk percentage for gc
health_disk_usage_gc_threshold=70               # agent disk percentage threshold for gc
health_disk_usage_percent_threshold=90          # max disk usage percentage
keep_past_manifests=3                           # how many past manifest to keep
monitor_check_timeinterval=120                  # application monitoring interval
monitor_probation_enabled=true                  # probation for application monitoring script
```

```
monitor_publish_enabled=false                # whether to publish application monitoring data
packageMgr_garbage_freq=1800                 # agent gc interval in seconds
packageMgr_install_package_age=43200         # agent package gc max age for installed package
packageMgr_install_package_min_age=7200      # agent package gc min age for installed package (sho
packageMgr_package_age=86400                 # agent package gc max age for downloaded package
packageMgr_package_min_age=7200              # agent package gc min age for downloaded package (sh
pkiauth_enabled=false                        # pki auth
safeshutdown_busy_check_timeout=86400        # timeout checking for safe shutdown
skip_cleanup_on_failure=false                # skip cleanup when deploy application failure
system_restart_time_threshold=300            # system uptime check for system reboot
threadMgr_garbage_freq=60                    # agent thread gc interval
threadMgr_thread_age=3600                    # agent thread age for gc
```

Override Configuration

```
curl -k -X POST -d '{"configs": {"<configName>": "<configValue>"}}' https://host:12020/agent/config
```