**Deployment Guide Portal Microservices & Consul**

Table of Contents

[**Revision History** 1](#_Toc472597705)

[**About this guide** 2](#_Toc472597706)

[**Key terms** 2](#_Toc472597707)

[**High-level Architectural Diagram** 3](#_Toc472597708)

[**Deploying Microservices** 3](#_Toc472597709)

[How to deploy widget-ms 3](#_Toc472597710)

[How to install widget-ms 4](#_Toc472597711)

[**Deploying Consul** 5](#_Toc472597712)

[Key Agent Files 5](#_Toc472597713)

[About config.json.<env> files 5](#_Toc472597714)

[About widgets-service.json file (Services JSON) 6](#_Toc472597715)

[Build Consul Agents. 7](#_Toc472597716)

[Deploy / Installing Consul Agents 8](#_Toc472597717)

[**Contact:** 9](#_Toc472597718)

# **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Revision | Author | Date | Status |
| 0.1 | Ikram Ikramullah (fi241c@att.com) | 01/19/2017 | Initial Draft |
|  |  |  |  |
|  |  |  |  |

# **About this guide**

This guide is to help deployers of various environments to deploy Portal Microservices and Consul. Consul is a system to help applications register and discover services so that service locations are not bound to the clients that would like to interact with it.

This guide is not a guide or explanation of Consul – We will cover things strictly from ECOMP Portal’s deployment needs. For details on Consul – please visit <https://www.consul.io/>.

# **Key terms**

1. Consul: System that provides service registration and discovery. All agents and their communication protocols can be jointly be referred to as Consul.
2. Consul Cluster: The set of VMs running Consul
3. Consul Server Agent: A Consul Agent run with config.json file dictating “server”: true – See below for more details.
4. Consul Client Agent: A Consul Agent run with config.json file dictating “server”: false – See below for more details.
5. Service / Microservice: Microservice or service from ECOMP Portal’s perspective is an independent http server process running on a location known only to Consul, not service’s clients – for example, ECOMP Portal Web Application.
6. Consul Servers Leader: Each Cluster has a Leader chosen using the list of Server Agents by Consul. Leader election is a process that is internal to consul – Deployers have no control over that. However, when should the election start, can be controlled using boostrap\_expect parameter configuration explained below.
7. Consul Servers Followers: All Server agents that have joined the cluster but were not elected leaders, are follower agents. A follower agent collects from its leader all data in form of logs so that at any point if the current leader vanishes, a follower can take control (using same election process) and start from the same state the previous leader left the cluster.
8. Gossip Protocol: Consul makes the agents use a protocol it calls Gossip to figure out all the active, inactive or failed nodes and services in the cluster.

# **High-level Architectural Diagram**

Following is a high-level architectural diagram of Portal Microservices Architecture with Consul.

FE-VM1

Consul Server1

(172.20.20.10)

Leader

BE-VM2

Consul Client2

(172.20.20.14)

**widgets-service** 172.20.20.14:8083/widgets

**portal-be**

(discover services & delegate)

FE-VM2

Consul Server2

(172.20.20.11)

Follower

ADMIN-VM

Consul Server3

(172.20.20.12)

Follower

BE-VM

Consul Client1

(172.20.20.13)

**widgets-service** 172.20.20.13:8083/widgets

**portal-be**

(discover services & delegate)

The first three orange boxes in the first column are depict VMs that are running Consul Servers – one of which is the leader – Non leaders receive logs for the rainy day when they have to become leader for some reason (for example, if the current leader becomes unavailable). The second two orange boxes depict Consul Client Agents looking out for our services’ health etc..

Note also that the number of Consul Servers can vary from environment-to-environment. The diagram shows what’s expected in Production and ETE environments – 3 Server Agents and 2 Client Agents. However, in QA and Devn1 etc., if a third VM is not available (which is the case in Devn1 and QA at the time of this writing), that shouldn’t affect how Consul behaves. It is just that, if two, the server leader agent election will be between two contenders instead of three.

# **Deploying Microservices**

Currently, the following Microservices are supported by ECOMP Portal

1. widgets-ms: This service is responsible managing widgets in ECOMP Portal and serving them to dashboards and catalogs in Portal.

Following is how to deploy the service.

## How to deploy widget-ms

Go to the following Jenkins build

<http://sdt-ecomp-portal.vci.att.com:25927/jenkins/view/Portal_Microservices/job/WidgetMS/>

Click on the “Build with Parameters” – Use master branch (the default value) and click on Build. Console Output can be seen to

Build will take about a minute – See if the final message is Success.

Copy the version number – For example 1702.3.44 – This you will need in the next step.

## How to install widget-ms

Go to the following Jenkins for installing the service – This is the same as Portal’s Web Application – Only the parameters will change.

<http://sdt-ecomp-portal.vci.att.com:25927/jenkins/view/Portal_Microservices/job/ECOMP-CD-SWM/>

A sample build will look something like the following – Note: The password hash should be created using instructions right below the Password field in the form.



As you can see, we are choose the BE servers the install of the widget-ms, telling the component is WidgetMS, the correct version we got from the build process above, ATTUID of the person and the password hash (see how it is generated in the instructions right below the field).

Press Build button now and go to Consol Output to see the install progress.

After Successful install, the service’s older (if any) instance will be stoped, replaced with the new version and restarted.

The following command on the target VMs (any of the two BEs we used) should show a single instance of the widget-ms running.

>> ps –ef | grep widget

# **Deploying Consul**

## Key Agent Files

Consul agents need new building only if one or more of the following have changed.

1. config.json.<env> files (needed for both client and server agents)
2. widgets-service.json file (needed only for client agents)

Now a quick explanation of each.

### About config.json.<env> files

Some key facts about these files.

The number of these files for one lab/environment will be the same as the number of agents. So, referring back to architectural diagram above, it will be five – 3 Server Agents and 2 Clients Agents.

1. As explained in the Architecture section, the number of servers can be less than 3 for some environments.
2. Following is a sample config.json (followed by explanation of key parameters). This file is for a server agent designed to run on FE server in Devn1 environment.

{

"bind\_addr": "135.21.155.75",

"datacenter": "dc-devn1-middletown",

"data\_dir": "/opt/app/MSCommon/consul/data/",

"log\_level": "INFO",

"enable\_syslog": true,

"enable\_debug": true,

"node\_name": "mtanjv9porf01-eth1.aic.cip.att.com",

"server": true,

"bootstrap\_expect": 1,

"leave\_on\_terminate": false,

"skip\_leave\_on\_interrupt": true,

"rejoin\_after\_leave": true,

"retry\_join": [

"135.21.155.76:8301",

"192.169.3.77:8301",

"192.169.3.78:8301"

]

}

The above config files tells the following about this agent and the cluster (fields explained based on most editable / importance)

* 1. The deployer wants this agent to bind to (essentially meaning ‘interact with’ other agents) using IP 135.21.155.75 (“bind\_addr”)
  2. The deployer wants it to be a server agent (“server” : true).
  3. The deployer wants to start the leader election process as soon as there is 1 server agent available (“bootstrap\_expect” : 1) – Note, this does not mean that there will be just one server agents allowed – Any number of server agents can be part of the cluster – the flag only tells Consul when the election process should start for the leader (and hence the cluster is functional).
  4. “node\_name” parameter should be the one using which the services will talk to each other – Note –eth1 value. That’s the interface against which certificates were generated. If not eth1 (or whatever is the correct one for each VM) is omitted, there will be certificate issues and services won’t discover each other.
  5. “data\_dir”: is the place where agents keep their persistence data stored.

The above was description of server agent’s config.json – The following is one for client agent – Most of the parameters are self-explanatory – But the key differences are

1. server flag is false – meaning it’s a client agent looking after actual services.
2. bootstrap\_expect flag is missing – Clients do not elect or participate in leader election – That flag is not needed.

Client config.json example.

{

"bind\_addr": "192.169.3.77",

"datacenter": "dc-devn1-middletown",

"data\_dir": "/opt/app/MSCommon/consul/data/",

"log\_level": "INFO",

"enable\_syslog": true,

"enable\_debug": true,

"node\_name": "mtanjv9porb01-eth1.aic.cip.att.com",

"server": false,

"leave\_on\_terminate": false,

"skip\_leave\_on\_interrupt": true,

"rejoin\_after\_leave": true,

"retry\_join": [

"135.21.155.75:8301",

"135.21.155.76:8301",

"192.169.3.78:8301"

]

}

### About widgets-service.json file (Services JSON)

Client agents, apart from config.json file that tells them how they should run, also need to have service JSONs for services (that are running locally on that VM) for things service availability, health checks etc.

{

"service": {

"name": *"widgets-service"*,

"tags": [*"widgets"*],

"port": 8082,

"check": {

"id": *"widget-health-api"*,

"name": *"widgets service on port 8082/widgets"*,

"http": *"https://localhost:8082/widget/health"*,

"interval": *"10s"*,

"timeout": *"1s"*,

"tls\_skip\_verify":true

}

}

}

As clearly, we are telling the client agent to look out for a service running locally on port 8082 and report its stats to the server leader when required. The check object tells what method and endpoint to use to figure out the health of that service ([*https://localhost:8082/widget/health*](https://localhost:8082/widget/health)*)* and how often it should do that (internal) – Of course, every service it identifies, should be assigned a unique name.

An agent can look after multiple services running on that host/vm – When more services will come, their service JSONs will be dropped in the same (config) directory of the agent like widgets-service.json.

## Build Consul Agents.

Consul Agent build is only necessary if either config.json or widgets-service.json files have changed. All these changes must be checked in to the following CodeCloud Repo - master branch.

<https://codecloud.web.att.com/projects/ST_QUANTUM/repos/portal-microservices/browse/SWM/Consul/Linux>

After all files have been changed, committed and pushed, use the following steps to build and deploy.

Following is the build location for Agents.

<http://sdt-ecomp-portal.vci.att.com:25927/jenkins/view/Portal_Microservices/job/ECOMP-SupportingSoftware-SWM/>

Use the following parameters to build.



After the build, note down the build number to use for the install in next step.

## Deploy / Installing Consul Agents



The deployer here must have 3 config.json files for server agents and 2 for client agents and checked in to CodeCloud repo mentioned above.

After a few minutes, consul agents must have been restarted on all the VMs chosen above.

The following command on each VM must give one running instance of the consul agent.

>> ps –ef | grep consul

# **Contact:**

Forward your questions to the following individuals.

Ikram Ikramullah (fi241c@att.com)

Nicholas Picciocca (np4285@att.com)