Luke Stanish, Tom Boudwin, Walter Snyder, Christopher Ariza

Dr. Ngo

CSC 496

5/10/19

Project Deliverable 3

Zun is a container management service. By utilizing OpenStack and Cloudlab, Zun can use Docker containers to replace the need for virtual machines on OpenStack by using the API service to manage running applications without the need to manage clusters or servers.

To begin, spin up a default OpenStack profile on Cloudlab. Cloudlab's latest version is Rocky, so the Zun installation guide for OpenStack will also use the Rocky version. Zun and OpenStack both are actually a full version ahead of Cloudlab (Stein) but the version matching Cloudlab must be used. Zun requires two nodes to run a container, a controller node and a compute node.

The controller node runs the Identity service, Image service, management portions of Zun, management portion of Networking, various Networking agents, and the Dashboard. It also includes supporting services such as an SQL database, message queue, and Network Time Protocol (NTP). Optionally, the controller node runs portions of the Block Storage, Object Storage, and Orchestration services. The controller handles networking because it shares a name with the network node. By default, this makes the controller node subsume the responsibilities of the network node. The controller node requires a minimum of two network interfaces.

The compute node runs the engine portion of Zun that operates containers. By default, Zun uses Docker as container engine. The compute node also runs a Networking service agent that connects containers to virtual networks and provides firewalling services to instances via security groups. You can deploy more than one compute node. Each node requires a minimum of two network interfaces.

Of great importance are the configuration files for both nodes. Each node has a file which controls how it communicates and implements the various services of OpenStack. The compute node knows the API to use the various services on the controller. These files must be edited to reflect the unique instance when it is initiated. The various values can be found by combing the setup-controller.log file generated by each unique instance. Details will be provided later.

First, access the controller node by SSH. Before Zun can even begin to install here, there must be a created database, service credentials, and API endpoints. Use MySQL to connect to the database server as the root user. To create the service credentials, create a Zun user with an appropriate password, add admin role to that user, and create the Zun service entities. Then create the container service API endpoints. This sets the address and port number for communication with the compute node later on. This includes the public, internal, and admin endpoints. The previous step involving the Zun user simply created the service credentials for an actual Zun user which now needs to be created along with necessary directories. Install zun by cloning the Zun repository using Git and installing it using pip. Once again, it is of utmost importance that the version of Zun cloned matches the version of OpenStack, which is to say what Cloudlab uses when launching an OpenStack instance. A sample configuration file needs to be generated. Upon viewing the file, it is obvious that it communicates with the various services of OpenStack. This is the file that will be edited with the generated values for a particular instance. OSLO As mentioned before, these values come from the setup-controller.log file. This is a massive file, and can be hard to search after using nano setup-controller.log. To make life easier, push the .log file onto a repository which can be easily accessed. Then, use the find command to quickly locate the needed values. The RabbitMQ URL is the main cause of concern here. The MGMTIP, the IP address of the main interface, along with various auth ports used for the Keystone service, seem to stay relatively the same throughout multiple instances, but the RabbitMQ, which is a messaging service broker, had a unique URL each time.

The next step is to populate the Zun database. To finalize the installation, create zun-api.service and zun-wsproxy.service as upstart configs. Zun-api is an OpenStack-native REST API that processes API requests by sending them to the zun-compute over Remote Procedure Call (RPC). Zun-wsproxy provides a proxy for accessing running containers through a websocket connection. Zun-compute will be discussed in conjunction with the compute node. Start and enable the zun-api and zun-wsproxy and check their status to ensure that they are running. Before moving onto the compute node, ensure that the etcd component is also on the controller node. Etcd is a distributed key value store that provides a reliable way to store data across a cluster of machines. Rocky has this already installed, but earlier versions such as Queens do not. If an install is needed, the etcd config file must be edited to used the MGMTIP of the controller node. This enables access from other nodes via the controller node.

Next up is the compute node. As mentioned, this operates containers. Before installing Zun here, both Docker, a container service, and kuryr-libnetwork, Kuryr’s Docker libnetwork driver that uses Neutron to provide networking services and it provides containerised images for the common Neutron plugins. The Docker install requires no special values. Simply follow the installation guide and use the latest version of Docker instead of selecting a specific one. Kuryr follows in much the same fashion. Kuryr is on both the controller and compute node. Care must be taken to install it on both. This creates the network that is needed. There will be a user created for Kuryr, though not for Docker. This user also needs service credentials and admin rights as done for the Zun user in the controller node.

After those components are installed, the compute node must be prepared in a similar fashion to the installation of Zun on the controller node. Once again, a Zun user and the associated directories must be created. Clone and install Zun using Git and pip. Keep the correct version in mind. Though this has been stressed multiple times in this paper, it cannot be said enough. Generate a sample configuration file and use the same values to communicate with the API endpoints on the controller node. Configure sudoers for Zun users which allows any user added to the Zun database to interact with it. Kuryr and Docker also need to be configured. In Docker's case, the docker.service.d file must be edited to listen to a certain port and have etcd as the storage backend. Kuryr.conf must be edited to set its capability scope to global. These are standard, and require no further research. Etcd must be previously installed to allow it to operate as the storage backend. All that remains here is to create the zun-compute service. Zun-compute is a worker daemon that creates and terminates containers through Docker API Manage containers and compute resources in local host. It should be noted that the services created are used through the Zun user, which is the reason for its creation. Restart and enable zun-compute and check its status to ensure installation. After this, a container should be able to be launched on any network available to the controller node (can be found by neutron netlist on the controller node).

When launching a container, an error was received. Essentially, a host could not be identified and the launcher asked if the proper service was specified. When the initial success was had (at least in creating a container even with failures), versions had been accounted for. Initial roadblocks had been based around that issue. Now, it seems to be a problem with networking. By using the default OpenStack profile, the network configuration seems to be skewed. Two networks that should appear when running neutron netlist should be selfservice and provider. However these do not appear. When viewing the network topology, a gateway also seems to be down. A possible workaround could be in launching 3 virtual machines that would emulate a network node, a controller node, and a compute node. While this would be nice as a proof of concept, it would not complete the goal of full automation of the process using Cloudlab and OpenStack. Speaking of automation, there were many problems. First, automating a process that does not work manually seems pointless. There also were problems with the time of instantiating a branch that used automation. When spinning up an instance, it could take hours and would not even fully load. It seems the only way to configure Zun on OpenStack is to build an OpenStack instance from scratch. This way, it can be ensured that all the appropriate API endpoints and services are installed, updated, and correctly logged.