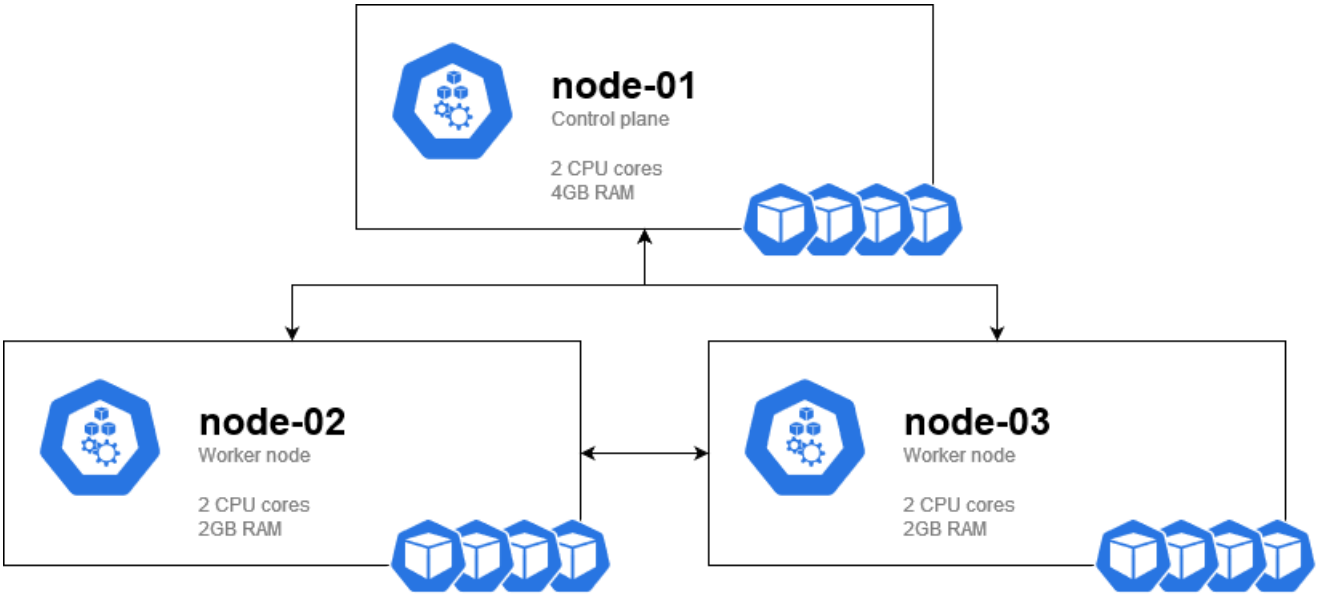
Installation guide



In this guest lecture, we'll build a 3-node Kubernetes cluster in an automated manner. In order to automate the set up of the cluster, we'll use a preconfigured ISO and an Ansible playbook. For the OS, we'll be using Fedora CoreOS which is preconfigured using Butane and Ignition to automatically create some users, install packages and configure settings. Curious how we prepared the image? Check out the bonus section at the end of this document!

1. Creating a virtual machine for each node

Before you begin the installation process, the cluster environment needs to be set up properly. Our cluster consists of three virtual machines, each with a specific role: one master node and two worker nodes. Creating the virtual machines can be done by executing the following steps:

1. Navigate to the course's folder and create a folder with your name
2. Select "New Virtual Machine..."
    1. With "Create a new virtual machine" selected, press "Next"
    2. Give the virtual machine a name (e.g. node-01), select your personal folder and press "Next"
    3. With the correct compute resource selected, press "Next"
    4. Select the Datastore you want to use and press "Next"
    5. With the latest version selected in the compatibility dropdown, press "Next"
    6. Configure the following settings on the "Select a guest OS" screen and press "Next":
        1. Guest OS Family: Linux
        2. Guest OS Version: Red Hat Fedora (64-bit)
    7. Customize the virtual machine's hardware and press "Next":
        1. CPU: 2
        2. Memory: 4 GB for the master node, 2 GB for the worker nodes
        3. Hard disk: 24 GB
        4. Network: your personal _pub network
        5. CD/DVD Drive: Datastore ISO file (select the fcos-38.iso)
    8. Review the settings and press "Finish"
3. Repeat step 2 for the remaining nodes

2. Installing Fedora CoreOS on the virtual machines

This is the part where you should just be able to sit back, grab a coffee and watch the magic happen, but we'll have to make sure that all the steps have completed before continuing to the next step.

So, start your virtual machines and wait for them to start up with the Fedora CoreOS ISO image. (You should see a screen like in the screenshot below)

Once the machine has booted, it'll start copying the files to the boot disk and show you a login screen. Don't log in yet! In the background, it's still installing some packages. After 2 reboots (so 3 boots in total, including the initial boot after the installation), you should be presented with a screen like in the screenshot below. You're now ready to move to the next step.

3. Connecting via SSH to virtual machines

We'll be using an Ansible playbook to create the Kubernetes cluster. Ansible will be connecting over SSH to the virtual machines. Before we can execute ansible, we need to add the virtual machines to our machine's `known_hosts` file.

To do this, we can use the `ssh-keyscan` command. Execute the following command for each host (from your local machine) to add each hosts' fingerprint to your `known_hosts` file.
`ssh-keyscan NODE_IP_ADDRESS >> ~/.ssh/known_hosts`

4. Creating an inventory file for Ansible

In order to tell Ansible which hosts to use while executing the playbook, we need to create an inventory file. First, copy the Ansible playbook directory (called `ansible_playbook`) from the USB to your local filesystem. After that, copy the baked-in SSH key (provided on the USB in the `fedora_coreos` directory) to your local machine. Once that's done, create an `inventory.yaml` file in the Ansible playbook directory on your local machine. Now, copy the example below to that file and adjust it to match your virtual machines' IP addresses. Make sure you also edit the `ansible_ssh_private_key_file` entries to match the SSH key file location on your machine.

```yaml
master:
  hosts:
    node-01:
      ansible_host: 1.1.1.1
      ansible_user: robot
      ansible_ssh_private_key_file: ./ssh_key
worker:
  hosts:
    node-02:
      ansible_host: 2.2.2.2
      ansible_user: robot
      ansible_ssh_private_key_file: ./ssh_key
    node-03:
      ansible_host: 3.3.3.3
      ansible_user: robot
      ansible_ssh_private_key_file: ./ssh_key
```

Verify that you can now connect to the machines using Ansible by executing the `ansible -m ping all -i inventory.yaml` command. If everything is set up correctly, you should see a response like this:

```
❯ ansible -m ping all -i inventory.yaml
node-01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
node-03 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
node-02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

5. Executing the Ansible playbook

Head over to the Ansible playbook directory on your local machine (you should have copied it from the USB in the last step). You should be able to see a `automated-install.yaml` file in this directory. This is the Ansible playbook which contains the references to the various roles that need to be executed on the nodes.

- **common role**: This is the role used to set the node's hostname (based on the inventory file) and install the required Kubernetes packages (kubelet, kubeadm and cri-o).
- **kubernetes-master role**: Using this role, the cluster gets initialized on the master node and some required components such as Flannel get installed.
- **kubernetes-worker role**: This role joins a worker node to the Kubernetes cluster.

The Ansible playbook requires the `community_general` Ansible collection to be present on the system. In order to install this, just execute `ansible-galaxy collection install community.general` in the `ansible_playbook` directory.

Executing the Ansible playbook - and thus creating the Kubernetes cluster - is as easy as executing the `ansible-playbook` command with our inventory file and the playbook file as arguments. In this case, your command should look something like this: `ansible-playbook -i inventory.yaml automated-install.yaml`. Once executed, you should be greeted by a screen like in the screenshot below.

```
❯ ansible-playbook -i inventory.yaml automated-install.yaml
[WARNING]: Collection community.general does not support Ansible version 2.10.8

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
ok: [node-02]
ok: [node-03]
ok: [node-01]
```

You can now let the Ansible playbook do its thing and watch how the cluster gets set up. When everything has successfully been executed (this might take a couple of minutes), you should see a message like this:

```
TASK [kubernetes-worker : Check if node has joined cluster] ********************************************************
ok: [node-02]
ok: [node-03]

TASK [kubernetes-worker : Join the Kubernetes cluster] *************************************************************
changed: [node-02] => (item=node-01)
changed: [node-03] => (item=node-01)

PLAY RECAP *********************************************************************************************************
node-01                    : ok=18   changed=13   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

node-02                    : ok=9    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

The Kubernetes cluster should now be initialized. This can be double checked by SSH'ing into the master node (node-01) with the ssh key and the "robot" user and executing the `kubectl get nodes` command as root.

```
❯ ssh robot@10.252.252.11 -i ssh_key
Fedora CoreOS 38.20231002.3.1
Tracker: https://github.com/coreos/fedora-coreos-tracker
Discuss: https://discussion.fedoraproject.org/tag/coreos

Last login: Thu Oct 26 13:34:53 2023 from 192.168.1.51
[robot@node-01 ~]$ sudo su
[root@node-01 robot]# kubectl get nodes
NAME      STATUS   ROLES           AGE     VERSION
node-01   Ready    control-plane   7m33s   v1.27.3
node-02   Ready    <none>          6m40s   v1.27.3
node-03   Ready    <none>          6m37s   v1.27.3
[root@node-01 robot]#
```

6. Copying the kubeconfig file to your local machine

First, make sure you have kubectl installed on your local machine. If you don't have it installed, follow this guide to do so. Once you've installed kubectl, create a ".kube" directory in your home directory (`mkdir ~/.kube`), this is the location where your kubeconfig file will sit.

While logged into the master node, copy the "/root/.kube/config" file to your the "robot" user's home directory (`sudo cp /root/.kube/config /var/home/robot/config`). Now, log out of the machine and copy the file to your local machine by using `scp -i ssh_key robot@MASTER_NODE_IP_ADDRESS:~/config ~/.kube/config` . Verify that the copy went correctly by checking the contents of the file: `cat ~/.kube/config` .

7. Verifying your connection to the cluster

Now that you have the kubeconfig file on your local machine, it's time to check if you can connect to the cluster. This can be done by executing the `kubectl get svc` command. The output of this command should look something like in the screenshot below:

```
❯ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   27m
```

If your output looks like the one shown in the screenshot, the connection to the cluster works and you should be able to continue to the next part of this guest lecture. If it doesn't, check your connection to the master server, verify that the kubelet service is running on the master node and double check that the Kubernetes API server pod is working.

8. Making sure that DNS resolving works

Sometimes CoreDNS doesn't use the `/etc/resolv.conf` on the node for DNS resolving. In order to fix this, you can restart the CoreDNS deployment. This should force CoreDNS to check the `resolv.conf` file again and use it.

Restarting the deployment can be done by executing this command: `kubectl rollout restart deployment coredns -n kube-system` .

Bonus: Fedora CoreOS ISO

Fedora CoreOS will be the base of the Kubernetes cluster, it's an open-source operating system and part of the Fedora Project (which is a community effort sponsored by Red Hat themselves). The OS itself is designed for running containerized workloads and cloud-native applications, which makes it perfect for running our cluster.

One of the interesting features of Fedora CoreOS is that we can prepare an ISO image with some configurations. For example, you can create users, create files/directories, create systemctl services and so on.. In order to create an ISO image like this, we'll be using 2 tools called Butane and Ignition.

Butane and Ignition

Butane is a YAML-based configuration language used for provisioning Fedora CoreOS (and Red Hat CoreOS) images. It simplifies the process of creating and managing Ignition configurations. Ignition is the first process to run on a CoreOS machine and applies the desired system configuration.

Using Butane, we can preconfigure things such as disk partitioning, file system creation, user creation, network configuration, and more. All of this can be done in a declarative YAML format, making it easier to manage and version control compared to Ignition.

To use Butane, we first need to install the CLI tool. This can be done using DNF on Fedora machines: `dnf install -y butane`. If you're not running any Fedora(-based) OS, you can also download the binary from the Github releases page. After that, you can write a Butane configuration file that describes the desired system state, including the various configuration parameters. You can then pass this file to the Butane utility, which converts it into an Ignition configuration that can be applied during the system boot process.

Once you have your Butane configuration file ready, you can pass it to the butane command-line utility to generate the corresponding Ignition configuration: `butane butane_config.bu -o ignition.ign`.

The resulting ignition.json file can then be supplied to the CoreOS installation process, which will apply the configuration during the initial system boot. On the USB drive you've been handed at the start of the lab, you'll find a directory called "coreos_configuration" with an example configuration.

Preconfiguring the ISO image

Using the Butane configuration from the USB drive, we'll preconfigure the ISO to do the following on boot:

- Create a user which we can use to log onto the machine
- Add the CRI-O DNF module
- Add the Kubernetes YUM repository
- Prepare networking for Kubernetes

With the "butane_config.bu" file in your working directory, you can execute the `butane butane_config.bu -o ignition.ign` command. This should now output an "ignition.ign" file.

Once that's done, we can install the "CoreOS Installer" CLI tool, which will be used for baking our configuration into the ISO image. You can obtain the latest version of the binary from the project's Github release page. If you're running a Fedora(-based) operating system, you can also use the `dnf install coreos-installer` command. One other option for installing the CLI tool is by using the Cargo package manager; `cargo install coreos-installer`.

With the CoreOS Installer installed and the Ignition configuration file ready, it's time to package the configuration along with the Fedora CoreOS ISO into a new image. This ISO image will server as a self-contained package that automatically configures the system it's installed on. This can be done by executing the following command:

```
coreos-installer iso customize \
    --dest-device /dev/sda \
    --dest-ignition fcos.ign \ # Replace this value with the Ignition file
    --dest-console ttyS0,115200n8 \
    --dest-console tty0 \
    -o fcos-38-$(date +"%Y%m%d").iso \
    ./fedora-coreos-38.20230819.3.0-live.x86_64.iso # Replace this with the ISO you downloaded
```

Once the command has completed, you'll see an ISO image with the current date in your working directory. You can now boot off this image and get a preconfigured machine.