Canonical

Quick deployment guides

Canonical

Contents

1. Getting started with Landscape

See also: what-is-landscape

This tutorial guides you through the process of installing Landscape Server on a Multipass virtual machine, configuring it, registering a client instance to Landscape, and running a script on your client instance. At the end, you'll also be guided through how to teardown your environment.

This tutorial should take about 45 minutes to complete.

1.1. Prerequisites

There's a few things you'll need before starting this tutorial.

1.1.1. Hardware

You'll need a workstation with enough disk space (25G) and memory (5G RAM) available to create two Multipass VMs: one with 20G disk space and 4G RAM for Landscape Server and one with 5G disk space and 1G RAM for Landscape Client. Your workstation will be called your "host machine" throughout this tutorial.

1.1.2. Multipass

For this tutorial, Multipass needs to be installed on your host machine. To install Multipass, run the following in your terminal:

```
sudo snap install multipass
```

If you want to learn more about Multipass, see their installation guide[1] and full documentation[2].

1.1.3. Ubuntu Pro token

You'll need an Ubuntu Pro token[3]. Ubuntu Pro is free for up to 5 machines, and you can use the free version for this tutorial.

If you already have an Ubuntu Pro account, you can copy your token from your Ubuntu Pro dashboard[4].

If you don't have an Ubuntu Pro account, first sign up for a free personal Ubuntu Pro account[5], then copy your token from your Ubuntu Pro dashboard[6].

1.2. Create the virtual machines

Now, let's create the Multipass virtual machines (VMs). From the command line, run:

```
multipass launch noble --cpus 2 --memory 4G --disk 20G --name tutorial-landscape-
server-noble
```

(continues on next page)

[1] https://canonical.com/multipass/docs/install-multipass
[2] https://canonical.com/multipass/docs
[3] https://ubuntu.com/pro
[4] https://ubuntu.com/pro/dashboard
[5] https://ubuntu.com/pro
[6] https://ubuntu.com/pro/dashboard

(continued from previous page)

```
multipass launch jammy --cpus 1 --memory 1G --disk 5G --name tutorial-landscape-client-jammy
```

This step will take a few minutes to complete, but you'll receive progress updates in the command line. These commands create two VMs total: one for your Landscape Server and one for Landscape Client. Your Landscape Server will run on Ubuntu 24.04 Noble Numbat (the VM named `tutorial-landscape-server-noble`) and Landscape Client will run on Ubuntu 22.04 Jammy Jellyfish (the VM named `tutorial-landscape-client-jammy`).

You'll need the IP address of the `tutorial-landscape-server-noble` VM later, so let's get it now. Run the following on your host machine:

```
multipass info tutorial-landscape-server-noble
```

Copy the IP address from the output, saving it somewhere you can access later.

The full output should be similar to the details below.

```
Name:           tutorial-landscape-server-noble
State:          Running
Snapshots:      0
IPv4:           10.253.187.38
Release:        Ubuntu 24.04.1 LTS
Image hash:     28d2f9df3ac0 (Ubuntu 24.04 LTS)
CPU(s):         2
Load:           0.05 0.10 0.04
Disk usage:     1.8GiB out of 19.3GiB
Memory usage:   410.0MiB out of 3.8GiB
Mounts:         --
```

In this example output, the IP address for the `tutorial-landscape-server-noble` VM is `10.253.187.38`. We'll use that IP address throughout this tutorial, but the commands you run later should use the IP address that you just copied and saved.

1.3. Install Landscape Server

Now, we're ready to install Landscape Server! We'll install Landscape Server on the `tutorial-landscape-server-noble` VM. To do this, open a shell on that VM:

```
multipass shell tutorial-landscape-server-noble
```

After running that command, the prompt should change to `ubuntu@tutorial-landscape-server-noble`. This means you're now in your `tutorial-landscape-server-noble` VM, and any commands will be run on that VM instead of your host machine.

Now install some required packages by running the command below:

```
sudo apt update && sudo apt install -y ca-certificates software-properties-common
```

Then add the `landscape/latest-stable` PPA to get access to the Landscape Server packages by running this command:

```
sudo add-apt-repository -y ppa:landscape/latest-stable
```

Your VM's package information will now be up-to-date and includes information about the Landscape packages. Run the command below to install Landscape Server on your VM:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install -y landscape-server-quickstart
```

The installation will take some time during which you'll get a lot of output. You can ignore the output for this tutorial. For the full details on installing Landscape in quickstart mode, see how-to-quickstart-installation.

Once installation is complete, you can exit the shell by executing the `exit` command:

```
exit
```

You command prompt should go back to your standard host machine prompt after you exit the VM.

1.4. Register the first Landscape administrator

From a browser on your host machine, navigate to `https://10.253.187.38` replacing the IP address in the URL with the one for your Landscape Server VM. Your browser will likely warn about a self-signed certificate. It's OK to accept the risk and continue in this case. Complete the form to create the first admin user for the `standalone` account. Once you complete the form, Landscape will automatically bootstrap your new account. Now you are logged in to Landscape and can start registering and managing client computers.

1.5. Install and configure Landscape Client

From your host machine, open a shell to the `jammy` client VM.

```
multipass shell tutorial-landscape-client-jammy
```

Your prompt should now be `ubuntu@tutorial-landscape-client-jammy`.

Attach your pro token, replacing `your-pro-token` with your actual Pro token.

```
sudo pro attach your-pro-token
```

When completed, the output should include lines similar to the ones below (along with other lines that we are not displaying here):

```
This machine is now attached to 'Ubuntu Pro - free personal subscription'

     Account: your_name@example.com
Subscription: Ubuntu Pro - free personal subscription
```

Next, get the server's public SSL certificate and save it where `landscape-client` can use it when it needs to make HTTPS requests to your Landscape Server. Replace the IP address with the one for your Landscape Server VM, but keep the `:443` port.

```
echo | openssl s_client -connect 10.253.187.38:443 | openssl x509 | sudo tee /etc/
landscape/server.pem
```

Now you'll install Landscape Client.

```
sudo apt update && sudo apt install -y landscape-client
```

Edit the `/etc/landscape/client.conf` file to match the contents below. Be sure to replace the IP address in both `url` and `ping_url` with the one for your Landscape Server VM.

```
[client]
log_level = info
url = https://10.253.187.38/message-system
ping_url = http://10.253.187.38/ping
data_path = /var/lib/landscape/client
ssl_public_key = /etc/landscape/server.pem
account_name = standalone
computer_title = tutorial-landscape-client-jammy
include_manager_plugins = ScriptExecution
script_users = landscape,ubuntu
```

If you're not sure how to edit the file, you can use `nano` to do so.

```
sudo nano /etc/landscape/client.conf
```

Change the file as needed and press `CTRL-O` followed by `ENTER` to save the file and `CTRL-X` to exit.

Next, send a registration request to Landscape Server.

```
sudo landscape-config --silent
```

You should see a message indicating that the registration message was successfully sent to Landscape Server. If you get a message indicating that client was unable to connect to the server, double-check that you downloaded the server's certificate to the `/etc/landscape/` directory and that the IP address in the `/etc/landscape/client.conf` file is correct.

You can now exit your Jammy VM and return to your host machine.

```
exit
```

Go to your Landscape Server UI in your browser and click the arrow icon in the header. You should have a notification that a pending computer needs attention. Click that notification, select your computer, and click **Accept**.

1.6. Run a script

From the home page in the Landscape web portal, click on the **Scripts** tab. Click **Add script** to create a script with the following contents to be run as the `ubuntu` user. Give it the name `Hello World` and save it.

```
#!/bin/bash
echo "Hello, World!" > /home/ubuntu/hello
```

From the **Computers** page, select your computer and then click the **Scripts** tab. Select the `Hello World` script and click **Next**. You can leave all the other defaults, then click **Run** to create an activity to run the script on your Jammy VM.

Once the activity succeeds, the status of your activity in the web portal will change to "1 activity finished successfully". You can check the file was created by opening a shell on your

Jammy VM.

```
multipass shell tutorial-landscape-client-jammy
```

The directory listing should show the file `hello` in the `ubuntu` user's home directory.

```
ls -l
```

And the contents should be `Hello, World!`.

```
cat hello
```

After you've confirmed the file exists on your VM, you can return to your host machine:

```
exit
```

1.7. Cleanup

Congratulations! You now have successfully installed Landscape Server on a Multipass VM, registered another Multipass VM running Landscape Client, and executed a script on that VM. Feel free to explore the other management features that Landscape Server has to offer.

When you're done, don't forget to remove the Multipass VMs from your host machine:

```
multipass delete tutorial-landscape-server-noble
multipass delete tutorial-landscape-client-jammy
multipass purge
```

2. Getting started with MicroCeph

This tutorial will guide you through your first steps with MicroCeph. We will deploy a Ceph cluster on a single node using MicroCeph and store a JPEG image in an S3 bucket managed by MicroCeph.

2.1. How you'll do It

You will install MicroCeph, initialise the cluster, and add storage. Then, you will enable the S3-compatible Ceph Object Gateway (RGW) on your node and create an S3 bucket. Finally, you will upload an image to the bucket, consuming the storage via RGW.

As we progress, you will also interact with your cluster by checking its health, adding disks, and enabling RGW.

By the end of this tutorial, after successfully using MicroCeph to store an image, you will have a foundational understanding of how MicroCeph works, and be ready to explore more advanced use cases.

2.2. What you'll need

- The latest Ubuntu LTS version. Find Ubuntu release information here[7].
- 2 CPU cores
- 4 GiB RAM
- 12GiB disk space
- An Internet connection

2.3. Install MicroCeph

First, install MicroCeph as a snap package from the Snap Store:

```
sudo snap install microceph
```

Disable the default automatic Snap upgrades to prevent MicroCeph from being updated automatically:

```
sudo snap refresh --hold microceph
```

Caution:

Failing to set this option may result in unintended upgrades, which could critically impact your deployed cluster. To prevent this, all subsequent MicroCeph upgrades must be performed manually.

2.4. Initialise your cluster

Next, bootstrap your new Ceph storage cluster:

[7] https://ubuntu.com/about/release-cycle

```
sudo microceph cluster bootstrap
```

This process takes 3 to 5 seconds.

Check the cluster status:

```
sudo microceph status
```

The output should look somewhat as shown below:

```
user@host:~$

MicroCeph deployment summary:
- ubuntu (10.246.114.49)
 Services: mds, mgr, mon
    Disks: 0
```

Your cluster deployment summary contains your node's hostname (IP address). In our case, it's `ubuntu` (`10.246.114.49`), along with information about the services running and available storage. You'll notice that the cluster is healthy with one node and three services running, but no storage has been allocated yet.

Now that the cluster is initialised, we'll add some storage to the node.

2.5. Add storage

Let's add storage disk devices to the node.

We will use loop files, which are file-backed Object Storage Daemons (OSDs) convenient for setting up small test and development clusters. Three OSDs are required to form a minimal Ceph cluster.

Execute the following command:

```
sudo microceph disk add loop,4G,3
```

```
user@host:~$

+-----------+---------+
|   PATH    | STATUS  |
+-----------+---------+
| loop,4G,3 | Success |
+-----------+---------+
```

Success! You have added three OSDs with 4GiB storage to your node.

Recheck the cluster status:

```
sudo microceph status
```

```
user@host:~$
```

```
MicroCeph deployment summary:
- ubuntu (10.246.114.49)
Services: mds, mgr, mon, osd
Disks: 3
```

You have successfully deployed a Ceph cluster on a single node.

Remember that we had three services running when the cluster was bootstrapped. Note that we now have four services running, including the newly added `osd` service.

2.6. Enable RGW

As mentioned before, we will use the Ceph Object Gateway to interact with the object storage cluster we just deployed.

2.6.1. Enable the RGW daemon on your node

```
sudo microceph enable rgw
```

Note:

By default, the `rgw` service uses port 80, which may not always be available. If port 80 is occupied, you can specify an alternative port, such as 8080, by adding the `--port <port-number>` parameter.

Run the status check again to confirm that the `rgw` service is reflected in the status output.

```
sudo microceph status
```

```
user@host:~$

MicroCeph deployment summary:
- ubuntu (10.246.114.49)
Services: mds, mgr, mon, rgw, osd
Disks: 3
```

2.6.2. Create an RGW user

MicroCeph is packaged with the standard `radosgw-admin` tool that manages the `rgw` service and users. We will now use this tool to create an RGW user called `user`, with the display name `user`.

```
sudo radosgw-admin user create --uid=user --display-name=user
```

The output should include user details as shown below, with auto-generated access and secret keys.

```
user@host:~$
```

```
 {
"user_id": "user",
"display_name": "user",
"email": "",
"suspended": 0,
"max_buckets": 1000,
"subusers": [],
"keys": [
    {
        "user": "user",
        "access_key": "NJ7YZ3LYI45M4Q1A08OS",
        "secret_key": "H7OTclVbZIwhd2o0NLPu0D7Ass8ouSKmtSewuYwK",
        "active": true,
        "create_date": "2024-11-28T13:07:41.561437Z"
    }
],
...
```

2.6.3. Set user secrets

Let's define secrets for this user, setting `access_key` to `foo`, and `--secret-key` to `bar`.

```
sudo radosgw-admin key create --uid=user --key-type=s3 --access-key=foo --secret-key=bar
```

```
user@host:~$

 ...
 [
     {
         "user": "user",
         "access_key": "NJ7YZ3LYI45M4Q1A08OS",
         "secret_key": "H7OTclVbZIwhd2o0NLPu0D7Ass8ouSKmtSewuYwK",
         "active": true,
         "create_date": "2024-11-28T13:07:41.561437Z"
     },
     {
         "user": "user",
         "access_key": "foo",
         "secret_key": "bar",
         "active": true,
         "create_date": "2024-11-28T13:54:36.065214Z"
     }
 ],
...
```

2.7. Consuming the storage

2.7.1. Access RGW

Before attempting to consume the object storage in the cluster, validate that you can access RGW by running **curl** on your node.

Find the IP address of the node running the `rgw` service:

```
sudo microceph status
```

```
user@host:~$

MicroCeph deployment summary:
- ubuntu (10.246.114.49)
Services: mds, mgr, mon, rgw, osd
Disks: 3
```

Then, run **curl** from this node.

```
curl http://10.246.114.49
```

```
user@host:~$

<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult xmlns="http://
s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID></Owner><Buckets>
</Bucket
```

2.7.2. Create an S3 bucket

You have verified that your cluster is accessible via RGW. To interact with S3, we need to make sure that the `s3cmd` utility is installed and configured.

Install and configure `s3cmd`

To install `s3cmd`, run the following command:

```
sudo apt-get install s3cmd
```

To configure the s3cmd tool, create a file named `.s3cfg` in your home directory. This should be an INI-style configuration file with a single `[default]` section and key-value pairs for configuration.

Run the below command to create the file and configure s3cmd:

```
user@host:~$
```

```
cat > ~/.s3cfg <<EOF
[default]
access_key = foo
secret_key = bar
host_base = ubuntu
host_bucket = ubuntu/%(bucket)
check_ssl_certificate = False
check_ssl_hostname = False
use_https = False
EOF
```

Instead of running this command, you can of course also set up the configuration file using your favourite editor.

This configuration will do the following:

- Configure secret and access key that we had set earlier.
- Configure the host to contact. We have named our host `ubuntu`, so this is what we will set here.
- Configure the host bucket template. The host bucket scheme allows users to specify virtual hosting style access or other access modes. For our uses, we will set it to the host name, followed by the bucket name.
- Finally, we did not configure SSL/TLS for our endpoint, so we are disabling it for s3cmd as well.

As a good security practice, it should also be ensured that the `.s3cfg` file is only readable by the user as it does contain the secret key. Run chmod like this:

```
user@host:~$

chmod 0600 ~/.s3cfg
```

Create a bucket

You have verified that your cluster is accessible via RGW. Now, let's create a bucket using the `s3cmd` tool:

```
s3cmd mb -P s3://mybucket
```

Note:

The `-P` flag ensures that the bucket is publicly visible, enabling you to access stored objects easily via a public URL.

```
user@host:~$

Bucket 's3://mybucket/' created
```

Our bucket is successfully created.

2.7.3. Upload an image into the bucket

```
s3cmd put -P image.jpg s3://mybucket
```

```
user@host:~$

upload: 'image.jpg' -> 's3://mybucket/image.jpg'  [1 of 1]
66565 of 66565   100% in    0s     4.52 MB/s  done
Public URL of the object is: http://ubuntu/mybucket/image.jpg
```

The output shows that your image is stored in a publicly accessible S3 bucket. You can now click on the public object URL in the output to view the image in your browser.

2.8. Cleaning up resources

If you want to remove MicroCeph, you can purge the snap from your machine using:

```
sudo snap remove microceph --purge
```

This command stops all running services and removes the MicroCeph snap, along with your cluster and all its contained resources.

Note:

Note: the `--purge` flag will remove all persistent state associated with MicroCeph. The `--purge` flag deletes all files associated with the MicroCeph package, meaning it will remove the MicroCeph snap without saving any data snapshots. Running the command without this flag will not fully remove MicroCeph; the persistent state will remain intact.

Tip:

Skipping the **`purge`** option is useful if you intend to re-install MicroCeph, or move your configuration to a different system.

```
user@host:~$

2024-11-28T19:44:29+03:00 INFO Waiting for "snap.microceph.rgw.service" to
stop.
2024-11-28T19:45:00+03:00 INFO Waiting for "snap.microceph.mds.service" to
stop.
microceph removed
```

2.9. Next steps

You have deployed a healthy Ceph cluster on a single-node and enabled RGW on it. Even better, you have consumed the storage in that cluster by creating a bucket and storing an image object in it. Curious to see what else you can do with MicroCeph?

See our how-to guides, packed with instructions to help you achieve specific goals with MicroCeph.

Or, explore our Explanation and Reference sections for additional information and quick references.

3. Getting started with MAAS

This guide shows you how to install MAAS, set it up for either a Proof-of-Concept (POC) or a production environment, and verify that it is working.

3.1. Prerequisites

- A host running Ubuntu 22.04 LTS (Jammy) or newer.
- Administrative privileges (sudo) on the host.
- Network access to download snaps or packages.
- (Production only) A PostgreSQL server (version 14 or newer recommended).
- (Production only) A plan for DNS forwarder and DHCP scope.

3.2. Install MAAS

3.2.1. Option 1 – Snap (recommended)

```
sudo snap install --channel=<version>/stable maas
```

Replace `<version>` with the desired MAAS version (for example, `3.6`).

3.2.2. Option 2 – Debian packages

```
sudo apt-add-repository ppa:maas/<version>
sudo apt update
sudo apt -y install maas
```

3.3. Post-install setup

3.3.1. POC setup

Install the test database and initialize MAAS:

```
sudo snap install maas-test-db
maas init --help
```

Follow the prompts to configure the POC environment.

3.3.2. Production setup

1. Disable conflicting NTP services:

   ```
   sudo systemctl disable --now systemd-timesyncd
   ```

2. Install and configure PostgreSQL:

   ```
   sudo apt install -y postgresql
   sudo -i -u postgres psql -c "CREATE USER \"$DBUSER\" WITH ENCRYPTED PASSWORD
   '$DBPASS'"
   sudo -i -u postgres createdb -O "$DBUSER" "$DBNAME"
   ```

3. Edit PostgreSQL authentication: Add this line to `/etc/postgresql/14/main/pg_hba.conf`:

```
host    $DBNAME    $DBUSER    0/0     md5
```

4. Initialize MAAS with the database:

```
sudo maas init region+rack --database-uri "postgres://$DBUSER:$DBPASS@
$HOSTNAME/$DBNAME"
```

5. Create an admin user:

```
sudo maas createadmin --username=$PROFILE --email=$EMAIL_ADDRESS
```

3.4. Configure and start MAAS

3.4.1. Check MAAS service status

```
sudo maas status
```

Example:

```
bind9        RUNNING
dhcpd        STOPPED
postgresql   RUNNING
```

3.4.2. Web UI setup

1. Open: `http://<API_HOST>:5240/MAAS`
2. Log in with your admin credentials.
3. Configure:
   - DNS forwarder (e.g., `8.8.8.8`)
   - At least one Ubuntu LTS image
   - SSH key (Launchpad, GitHub, or upload from `~/.ssh/id_rsa.pub`)

3.4.3. CLI setup

1. Log in:

```
maas login $PROFILE $MAAS_URL $(cat api-key-file)
```

2. Configure DNS:

```
maas $PROFILE maas set-config name=upstream_dns value="8.8.8.8"
```

3. Add an SSH key (`$SSH_KEY` must be set to a valid SSH key):

```
maas $PROFILE sshkeys create "key=$SSH_KEY"
```

3.5. Enable DHCP

3.5.1. Web UI

- Go to Subnets > VLAN > Configure DHCP
- Select options
- Save and apply

3.5.2. CLI

Find the subnet CIDR and fabric you want using this expression:

```
maas $PROFILE subnets read | jq -r '
  ["subnet", "|", "fabric ID", "|", "gateway IP"],          # header

  (.[] | [ .cidr, "|", (.vlan.fabric_id|tostring), "|", .gateway_ip ]) #rows
  | @tsv
' | column -t
```

Find the precise name of the primary rack controller with this expression, which always finds the primary rack, regardless of how many racks are active:

```
maas $PROFILE rack-controllers read | jq -r '.[] | .interface_set[] | .vlan?.
primary_rack // empty'
```

Plug those values into the following commands to configure DHCP:

```
maas $PROFILE vlan update $FABRIC_ID untagged dhcp_on=True primary_rack=$PRIMARY_
RACK_CONTROLLER
maas $PROFILE subnet update $SUBNET_CIDR gateway_ip=$MY_GATEWAY
```

3.6. Upgrading MAAS

3.6.1. General steps

1. Backup your system and database.
2. Verify Ubuntu release (`lsb_release -a`). Upgrade to 22.04 Jammy or 24.04 Noble as required.
3. Verify PostgreSQL version (14 required, 16 recommended).
4. Upgrade rack nodes first, then region nodes.

3.6.2. Upgrade commands

- Snap

  ```
  sudo snap refresh maas --channel=<version>/stable
  ```

- Debian package (PPA)

  ```
  sudo apt-add-repository ppa:maas/<version>
  sudo apt update && sudo apt upgrade maas
  ```

3.6.3. Version-specific notes

- MAAS 3.6: PostgreSQL 14+ supported; PostgreSQL 16 recommended.
- MAAS 3.5: Requires PostgreSQL 14.
- MAAS 3.3: PostgreSQL 12 deprecated. Upgrade to 14 before proceeding.
- MAAS 2.8 or earlier: Full backup required. Fresh install recommended if upgrade fails.

3.7. Troubleshooting notes

- NTP conflicts:

```
sudo systemctl disable --now systemd-timesyncd
```

- BMC migration (3.3+): Ensure unique BMC IP/username/password combinations.

3.8. Verification

After installation or upgrade:

```
lsb_release -a   # Verify Ubuntu release
maas --version   # Verify MAAS version
sudo maas status # Verify services running
```

3.9. Related documentation

- About controllers[8]
- Back up MAAS[9]
- Networking in MAAS[10]

[8] https://canonical.com/maas/docs/about-controllers
[9] https://canonical.com/maas/docs/how-to-back-up-maas
[10] https://canonical.com/maas/docs/about-maas-networking

4. Getting started with MicroK8s

4.1. What you'll need

- An Ubuntu environment to run the commands (or another operating system which supports `snapd` - see the snapd documentation[11]). If you don't have a Linux machine, you can use Multipass (see Installing MicroK8s with Multipass[12]).
- MicroK8s runs in as little as 540MB of memory, but to accommodate workloads, we recommend a system with at least 20G of disk space and 4G of memory.
- An internet connection

 Note: If you don't meet these requirements, there are additional ways of installing MicroK8s, including additional OS support and an offline deploy. See the *alternative install* (page **??**) page for details.

1. **Install MicroK8s**

   MicroK8s will install a minimal, lightweight Kubernetes you can run and use on practically any machine. It can be installed with a snap:

   ```
   sudo snap install microk8s --classic --channel=1.33
   ```

   More about setting the channel (page **??**)

2. **Join the group**

   MicroK8s creates a group to enable seamless usage of commands which require admin privilege. To add your current user to the group and gain access to the .kube caching directory, run the following three commands: `sudo usermod -a -G microk8s $USER` `mkdir -p ~/.kube` `chmod 0700 ~/.kube`

   You will also need to re-enter the session for the group update to take place:

   ```
   su - $USER
   ```

3. **Check the status**

   MicroK8s has a built-in command to display its status. During installation you can use the **`--wait-ready`** flag to wait for the Kubernetes services to initialise:

   ```
   microk8s status --wait-ready
   ```

4. **Access Kubernetes**

   MicroK8s bundles its own version of **`kubectl`** for accessing Kubernetes. Use it to run commands to monitor and control your Kubernetes. For example, to view your node: `microk8s kubectl get nodes`

   …or to see the running services:

   ```
   microk8s kubectl get services
   ```

[11] https://snapcraft.io/docs/installing-snapd
[12] https://discuss.kubernetes.io/t/installing-microk8s-with-multipass/21544

MicroK8s uses a namespaced kubectl command to prevent conflicts with any existing installs of kubectl. If you don't have an existing install, it is easier to add an alias (append to **~/.bash_aliases**) like this:

```
alias kubectl='microk8s kubectl'
```

5. **Deploy an app**

   Of course, Kubernetes is meant for deploying apps and services. You can use the **kubectl** command to do that as with any Kubernetes. Try installing a demo app:

   ```
   microk8s kubectl create deployment nginx --image=nginx
   ```

   It may take a minute or two to install, but you can check the status:

   ```
   microk8s kubectl get pods
   ```

6. **Use add-ons**

   MicroK8s uses the minimum of components for a pure, lightweight Kubernetes. However, plenty of extra features are available with a few keystrokes using "add-ons" - prepackaged components that will provide extra capabilities for your Kubernetes, from simple DNS management to machine learning with Kubeflow!

   To start it is recommended to add DNS management to facilitate communication between services. For applications which need storage, the 'hostpath-storage' add-on provides directory space on the host. These are easy to set up:

   ```
   microk8s enable dns
   microk8s enable hostpath-storage
   ```

   See the full list of addons (page **??**)

7. **Starting and Stopping MicroK8s**

   MicroK8s will continue running until you decide to stop it. You can stop and start MicroK8s with these simple commands:

   ```
   microk8s stop
   ```

   … will stop MicroK8s and its services. You can start again any time by running:

   ```
   microk8s start
   ```

   Note that if you leave MicroK8s running, it will automatically restart after a reboot. If you don't want this to happen, simply remember to run `microk8s stop` before you power down.

- One node not enough? Try *setting up a MicroK8s cluster* (page **??**).
- Want to experiment with alpha releases of Kubernetes? *See the documentation on setting channels* (page **??**).
- Need to fiddle with the Kubernetes configuration? *Find out how to configure the Kubernetes services* (page **??**).
- Find out how to run MicroK8s on *Windows, macOS or a Raspberry Pi* (page **??**).
- Having problems? Check out our *troubleshooting section* (page **??**).
- Love MicroK8s? Want to contribute or suggest a feature? *Give us your feedback* (page **??**).

- Find out how Devs, DevOps and businesses **really** use Kubernetes: Read the latest industry report[13]

[13] https://juju.is/cloud-native-kubernetes-usage-report-2021