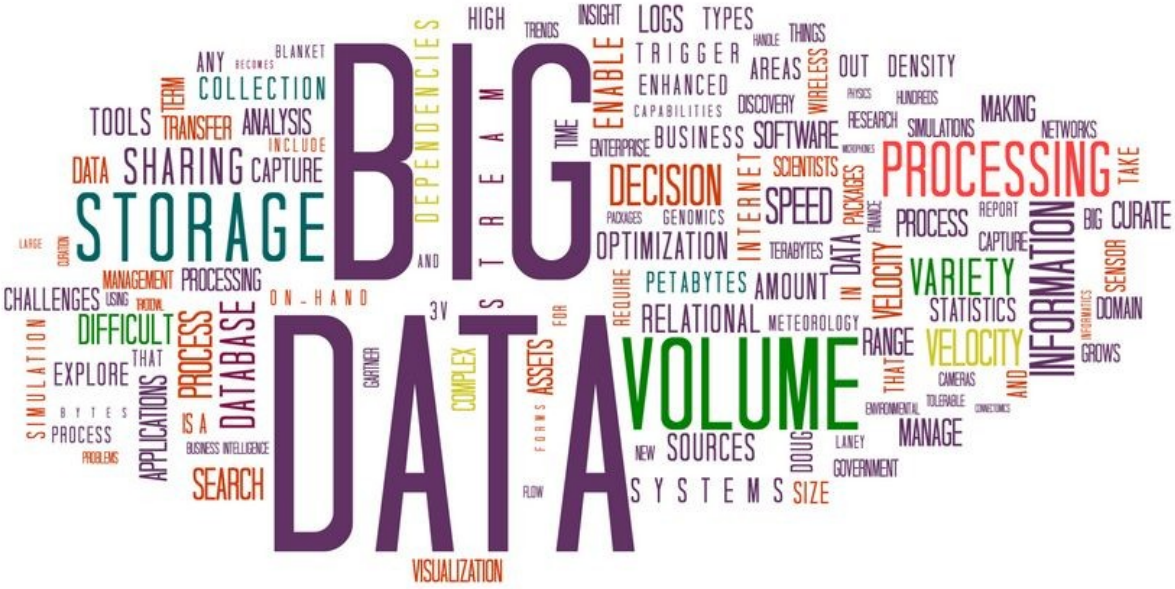
# Session 5

- Architecture
- OpenStack CLI
- Terraform
- Ansible
- Guest presentation:
  - Marc Gallofré Ocaña on big-data architecture for news
- Exercises 4 & 5



# OpenStack Command-Line Interface (CLI)

# OpenStack


- OpenStack:
  - framework for Infrastructure-as-a-Service (IaaS)
  - a «cloud operating system»
  - free, open-source
  - comprehensive, with components for delivering:
    - computing, networking and storage resources
    - identity and security
    - web UI, CLI, API

# OpenStack instances

- Launch instance:
  - flavor:
    - the size of a virtual machine and its characteristics
    - NREC has special high-performance flavours with GPUs
  - image: a file with a virtual disk that has a bootable OS installed on it
  - networks: IPv6, DualStack
  - security Groups: which ports to open to/from which addresses
  - keypair: public SSH keys
- Make snapshot:
  - provides a copy of a currently running VM or volume (virtual disk)
  - can be stored into and restored from an external service

# NREC sHPC instances

- Shared high-performance computing (sHPC) instances:
  - better processors
    - up to 64 processors 384Gb RAM
  - local hard drives
  - flavors for compute-heavy, memory-hungry and balanced workloads
  - isolated from the normal services.
  - much smaller overcommit ratios
  - scheduled downtime for maintenance
  - managed through OpenStack
    - must apply specially

# NREC vGPU instances (beta)

- Shared virtual GPU (vGPU) instances:
  - running on metal GPUs
    - Tesla v100 with 16Gb RAM
  - pre-built images
  - managed through OpenStack
  - must apply specially
    - for “pure” vGPU projects
    - vGPU resources must be used
    - delete instance when no longer needed.

# OpenStack command-line interface (CLI)

- More efficient than the dashboard
  - but a bit slow
  - we will use higher-level (scripted) Terraform instead
- Guide at https://docs.nrec.no/api.html#openstack-command-line-interface-cli
  - install:
    - sudo apt install python3-openstackclient
    - pip install python-openstackclient (perhaps in a virtual environment)
  - configure with either:
    - keystonerc.sh to set environment variables
    - ~/.config/openstack/clouds.yaml (needed by Terraform later)
  - we run it on «local machine» to control the «cluster machines»

# ./keystonerc.sh

```
export OS_USERNAME=your@nrec.log.in
export OS_PROJECT_NAME=uib-your-project
export OS_PASSWORD=sL6WUOiQb5R7fU0y
export OS_AUTH_URL=https://api.nrec.no:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_USER_DOMAIN_NAME=dataporten
export OS_PROJECT_DOMAIN_NAME=dataporten
export OS_REGION_NAME=bgo
export OS_INTERFACE=public
export OS_NO_CACHE=1

export OS_TENANT_NAME=$OS_PROJECT_NAME
```

- Pick up your password at
  - https://access.nrec.no
  - «Reset API password»
- Run with either

  source ./keystonerc.sh

  . ./keystonerc.sh
- Test with, e.g.,
  - openstack server list

# ~/.config/openstack/clouds.yaml

```
clouds:
  info319-cluster:
    auth:
      auth_url: https://api.nrec.no:5000/v3
      project_name: uib-your-project
      username: your@nrec.log.in
      password: sL6WUOiQb5R7fU0y
      user_domain_name: dataporten
      project_domain_name: dataporten
    identity_api_version: 3
    region_name: bgo
    interface: public
    operation_log:
      logging: TRUE
      file: openstackclient_admin.log
      level: info
```

- Pick up your password at
  - https://access.nrec.no
  - «Reset API password»
- Test with, e.g.,
  - openstack server list
- Also needed to run API
  - e.g., Terraform

# OpenStack CLI commands

- Overview commands:
  - openstack *resource_type* list
  - *resource_type* can be:
    - server
    - image
    - flavor
    - keypair
    - network
    - security group
    - and many others (--help)
  - example:
    - openstack server list
- Other usual commands:
  - openstack *res_type* create
  - openstack *res_id* show
  - openstack *res_id* delete
  - and many others
  - example:
    - openstack server \
      show spark-driver

# Terraform

# Terraform

- Terraform:
  - software tool for infrastructure management
  - free and open source
  - for cloud hosted or local (on-premise) virtual computing resources
  - provisioning and managing an infrastructure throughout its lifecycle
  - building, changing, and versioning resources safely and efficiently
  - based on scripts: human-readable configuration files
  - low-level components: compute, storage, and networking resources
  - high-level components: DNS entries, SaaS features, etc.
  - focus on (virtual) machine resources and OS-level configuration
  - less focus on applicaiton and middleware (→ Ansible does that)
  - interfaces for many infrastructure platforms, *including OpenStack*

# Terraform installation

- Example guide:
  - https://computingforgeeks.com/how-to-install-terraform-on-ubuntu/
- Configuration:
  - relies on a working OpenStack configuration
  - needs an OpenStack (or other) integration in info319-cluster.tf

# Terraform-OpenStack integration

In the Terraform configuration script (info319-cluster.tf)

```
# configure the OpenStack provider
terraform {
        required_version = ">= 1.0"
        required_providers {
                openstack = {
                        source = "terraform-provider-openstack/openstack"
                }
        }
}
# define the OpenStack cluster, either of
provider "openstack" {}                 # uses the OS_* environment variables
provider "openstack" {
        cloud  = "info319-cluster"      # defined in the ~/.config/openstack/clouds.yaml file
}
```

# Example configuration script

In the Terraform configuration script (info319-cluster.tf)

```
# ... after the OpenStack integration:
resource "openstack_compute_instance_v2" "info319-test" {
        name                = "info319-test"
        image_name          = "GOLD Ubuntu 22.04 LTS"
        flavor_name         = "m1.large"
        security_groups     =  ["default", "info319-spark-cluster"]
        key_pair                    = "info319-spark-cluster"
        network {
                name            = "dualStack"
        }
}
```

*Terraform uses names and ids from OpenStack!*

# Terraform resources

- Important resource types for OpenStack:
  - openstack_compute_instance_v2
    - a virtual machine with an OS image, a flavour, and a network
    - many other options: security group, keypair, etc.
  - openstack_compute_keypair_v2
    - create a keypair or import a public key
  - openstack_blockstorage_volume_v2
    - create or import a virtual disk with a size
  - openstack_compute_volume_attach_v2
    - a connection between a compute instance and a volume by their ids
  - lots of other resources
    - see https://registry.terraform.io/providers/terraform-provider-openstack/openstack/latest/docs
    - use the *Resources* tab to the left

# Terraform commands

- terraform init
  - set up initial Terraform state (terraform.tfstate)
- terraform plan
  - validate and dry-run infrastructure changes
- terraform apply
  - change infrastructure
- terraform destroy
  - tear down infrastructure (delete all the resources)
  - *...also destroys all data volumes (disks) permanently*
- terraform console (more later)
  - can be used to test terraform expressions interactively

# Terraform counts and variables

- Multiple resources in one declaration
- Example:

```
# spark worker volumes
resource "openstack_blockstorage_volume_v2" "terraform-worker-volumes"
{
    count       = 6
    name        = "terraform-worker-${count.index}-volume"
    size        = 100
}
```

# Terraform locals and modules

- Example:

```
locals {
        cluster_prefix      = "terraform-"
        worker_prefix       = "${local.cluster_prefix}worker-"
        num_workers         = 6
        volume_suffix       = "-volume"
}

# spark worker volumes
resource "openstack_blockstorage_volume_v2" "terraform-worker-volumes" {
        count           = local.num_workers
        name            = "${local.worker_prefix}${count.index}${local.volume_suffix}"
        size            = 100
}
```

# Terraform variables, loops, functions, and outputs

- List variables:
  - [ 0, 1, 2, 3, 4, 5, ]
  - [ for i in range(6): i ]
- Useful functions:
  - length(ipv6_string)
  - substr(ipv6_string, 1, string_length-2)
  - slice(some_list, start_idx, end_idx)
  - zipmap(a_list, another_list)
  - join("\n", list_of_lines)

- Map variables:
  - {2="B", 4="D", 5="E"}
  - zipmap(
    [2, 4, 5], ["B", "D", "E"]
    )
- Output variables:
  output "output_name" {
  value = *expression*
  }
- *Explore in the terraform console!*

# Terraform user-data.cfg

- How to configure instances beyond the OS image, network, security group, and keypair?
- OpenStack compute resources offer a user_data field:
  user_data = file("user-data.cfg")
- user_data is only for simple tasks
  - we will use Ansible for this later

- Example user-data.cfg file:

```
#cloud-config

apt_upgrade: true

packages:
- ansible
- emacs

power_state:
    delay: "+3"
    mode: reboot
```

*Do not turn upgrade on before you need it!*

# Terraform and SSH

- ~/.ssh/config on local machine need names and IPv6 addresses of cluster
  - generate host names from info319-cluster.tf :

    resource "local_file" "config-hosts-file" {

        content = "${local.config-hosts-string}\n"

        filename = "/YOUR/HOME/FOLDER/.ssh/config.terraform-hosts"

    }
  - add to ~/.ssh/config :

    Include ~/.ssh/config.terraform-hosts
    - ...remember the wildcard pattern spark.-*
  - use terraform console to explore expressions for config-hosts-string :

    > openstack_compute_instance_v2.info319-test

    > openstack_compute_instance_v2.info319-test.network[0]

# Ansible

# Ansible

- Ansible:
  - IT automation software tool written in Python
  - free, open source
  - configuring systems and deploying software
  - orchestrating advanced workflows:
    - support application deployment, system updates, and more
  - script and command-line based
  - simplicity and ease of use
  - strong focus on security and reliability
  - uses OpenSSH for transport (with alternatives)

# Ansible-Terraform integration

- Install Ansible locally, e.g.:
  - sudo apt install ansible
- Your «local-machine» Ansible needs to know the cluster hosts:
  - host names listed in /etc/ansible/hosts
  - generate host name list from info319-cluster.tf
- Your clusters hosts need to have Ansible installed already:
  - add to Terraform user-data.cfg :

```
packages:
- ansible
```

# Ansible test

- Check configuration:
  - (login to cluster manually first)
  - ansible all -m ping
- Test run:
  - ansible-playbook \
    info319-cluster.yaml
- Check mode:
  - ansible-playbook --check \
    info319-cluster.yaml

- Simple ./info319-cluster.yaml :

```
- name: Mark additions to .bashrc
  hosts: all
  tasks:

    - name: Add comment line to .bashrc
      ansible.builtin.lineinfile:
        path: .bashrc
        regexp: '^# Added by Ansible'
        line: "\n# Added by Ansible"
```

# Ansible playbooks

- Plays / blocks:
  - named
  - applies to one or more hosts
- Hosts:
  - all, host name, host-name pattern, host list, local machine
- Tasks:
  - a configuration/deployment/ orchestration on a host
  - realised as a module
  - provides module arguments
- Modules:
  - performs tasks on hosts

- Simple ./info319-cluster.yaml :

```
- name: Mark additions to .bashrc
  hosts: all
  tasks:

    - name: Add comment line to .bashrc
      ansible.builtin.lineinfile:
        path: .bashrc
        regexp: '^# Added by Ansible'
        line: "\n# Added by Ansible"
```

*Idempotency*:
executing an operation *multiple times* has the same effects as executing it *once*

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Mark additions to .bashrc
  hosts: all
  tasks:

    - name: Add comment line to .bashrc
      ansible.builtin.lineinfile:
        path: .bashrc
        regexp: '^# Added by Ansible'
        line: "\n# Added by Ansible"
```

Lists of collections and modules:
https://docs.ansible.com/ansible/latest/collections/index.html

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Installing Java
  gather_facts: false
  hosts: all
  tasks:

    - name: Install OpenJDK 8
      ansible.builtin.apt:
          name: openjdk-8-jdk-headless
      become: yes
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Installing Spark
  gather_facts: false
  hosts: all
  tasks:

    - name: Upload and unpack Spark archive
      ansible.builtin.unarchive:
        src: ./sources/spark-3.3.0-bin-hadoop3.tgz
        dest: /home/ubuntu
        remote_src: yes
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Installing Spark
  gather_facts: false
  hosts: all
  tasks:

    - name: Upload and unpack Spark archive
      ansible.builtin.unarchive:
        src: ./sources/spark-3.3.0-bin-hadoop3.tgz
        dest: /home/ubuntu

    - name: Symlink to Spark
      ansible.builtin.file:
        src: ./spark-3.3.0-bin-hadoop3
        dest: ./spark
        state: link
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Prepare Spark node SSH keys
  hosts: all
  tasks:

    ....upload and authorise the key...

    - name: Remove public Spark key file
      ansible.builtin.file:
        path: ./keys/NREC-INFO319-spark.pub
        state: absent
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Prepare Spark node SSH keys
  hosts: all
  tasks:

    - name: Upload private Spark key
      ansible.builtin.copy:
        src: /.../.ssh/info319-spark-cluster
        dest: ./.ssh/
        owner: ubuntu
        group: ubuntu
        mode: 0600
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Create Spark keypair
  hosts: 127.0.0.1
  connection: local
  tasks:

    - name: Create keypair for Spark
      community.crypto.openssh_keypair:
        path: ./keys/info319-spark-cluster
        size: 4096
        mode: 0600
        regenerate: always
```

- ...or reuse the keypair you already have

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Prepare Spark node SSH keys
  hosts: all
  tasks:

    ...

    - name: Authorise public Spark key
      ansible.posix.authorized_key:
        key: "{{ lookup('file',
'/PATH/keys/info319-spark-cluster.pub') }}"
        user: ubuntu
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- mount
- template
- command

```
- name: Prepare Spark node .ssh/config
  ...

    - name: Add ssh configuration for Spark
      community.general.ssh_config:
          host: "{{ item }}"
          hostname: "{{ lookup('file',
                  'ipv4-{{ item }}').split(' ')[1] }}"
          user: ubuntu
          port: 22
          identity_file:
/home/ubuntu/.ssh/info319-spark/cluster
          strict_host_key_checking: no
      with_items: "{{
                lookup('file', 'cluster-hosts')
                        .splitlines() }}"
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- parted, filesystem, mount
- template
- command

```
- name: Mount volume
  hosts: all
  become: yes
  tasks:

    - name: Create ext4 primary partition
      parted:
        device: /dev/sdb
        number: 1
        state: present

    - name: Create ext4 filesystem
      filesystem:
        dev: /dev/sdb1
        fstype: ext4
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- parted, filesystem, mount
- template
- command

```
- name: Mount volume
  hosts: all
  become: yes
  tasks:

    ....

    - name: Mount data_volume
      ansible.posix.mount:
        path: /home/ubuntu/volume
        src: /dev/sdb1
        fstype: ext4
        state: mounted
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- parted, filesystem, mount
- template
- command

```
- name: Installing HDFS

  ...

  - name: Upload Hadoop config files
    ansible.builtin.template:
      src: {{ item }}-site.xml.j2
      dest: /home/ubuntu/volume/hadoop
            /etc/hadoop/{{ item }}-site.xml
      owner: ubuntu
      group: ubuntu
      mode: 0644
    loop:
      - core
      - hdfs
      - mapred
      - yarn
```

# Ansible module examples

- lineinfile
- apt
- unarchive
- file
- copy
- openssh_keypair
- authorized_key
- ssh_config
- parted, filesystem, mount
- template
- command

```
- name: Prepare Hadoop namenode
  hosts: cluster-main
  tasks:

  ...

    - name: Start Hadoop/YARN cluster
      ansible.builtin.command:
        argv: |
            bash -c
                "if [[ $(jps) != *NameNode* ]];
                    then start-all.sh
                fi"
```

# Ansible loops

```
- name: Prepare Spark node .ssh/config
  ...

  - name: Add ssh configuration for Spark
    community.general.ssh_config:
        host: "{{ item }}"
        hostname: "{{ lookup('file',
                'ipv4-{{ item }}').split(' ')[1] }}"
        user: ubuntu
        port: 22
        identity_file:
/home/ubuntu/.ssh/info319-spark/cluster
        strict_host_key_checking: no
    with_items: "{{
                lookup('file', 'cluster-hosts')
                        .splitlines() }}"
```

```
- name: Installing HDFS
  ...

  - name: Upload Hadoop config files
    ansible.builtin.template:
        src: {{ item }}-site.xml.j2
        dest: /home/ubuntu/volume/hadoop
                /etc/hadoop/{{ item }}-site.xml
        owner: ubuntu
        group: ubuntu
        mode: 0644
    loop:
        - core
        - hdfs
        - mapred
        - yarn
```

# Ansible variables

```
- name: Install HDFS and YARN
  hosts: all
  tasks:

    - name: Register master_node expr
      shell: |
          grep terraform-driver /etc/hosts |
          cut -d' ' -f1
      register: master_node_expr

    - name: Set master_node fact
      set_fact:
        master_node: |
          "{{ master_node_expr.stdout }}"
```

```
- name: Install Zookeeper on the cluster
  hosts: all
  vars:
    num_zookeepers: 3
```

# Web UIs