Tampere University

COMP.SE.140: 21.11.2023

Tampere University

Course matters

- Information about the CI-capable GitLab is out
  - Please read carefully
  - Send questions
  - Next version TBD
- Grading of the communication exercise has been started
- Next weeks event is still under planning

Course status

SISU signups	108
Plus signups	150
Docker/compose exercise	86
Communication exercise	69
Ansible exercise	29
Project	1

Tampere University

Master thesis in a research project?

- We may have options starting ~January
- Possible topics
  - IoT and Edge computing
  - Data management systems
  - Declarative programming
  - Requirements traceability
- Expected competences
  - Strong software skills
  - Analythical thinking
  - Both independent and group work
  - Writing in English

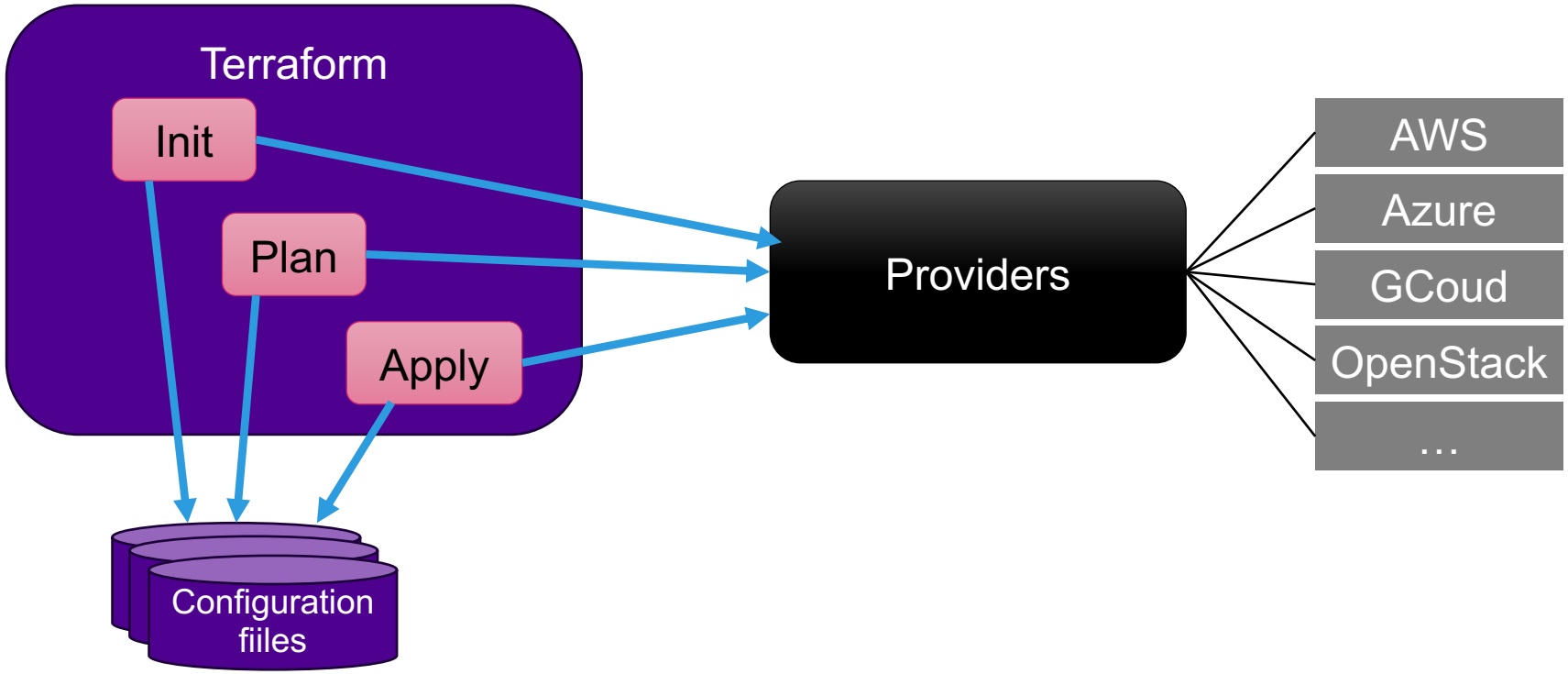
Automation includes

- Building
  -> no command-line tools needed
- Testing
  -> run frequently
- Other quality analysis
  -> less manual inspection needed;
- Deployment
  -> VMs and containers created automatically
  -> configuration management
- Database tools
  -> initialization
  -> management
- Scaling

Terraform

- An IaC tool to build infrastructure
  - Virtual machines
  - Storage
  - Network
- Declarative approach, too.
  - User writes a specification of the target state
- Pluggable architecture
  - Can have providers for AWS, Azure, gCloud, Open Stack …
- Code – (plan –)apply

Terraform


Terraform
Init
Plan
Apply
Configuration
fiiles
Providers
AWS
Azure
GCoud
OpenStack
…

Tampere University

example.tf

```
terraform {
    required_providers {
        aws = {
            source = "hashicorp/aws"
            version = "~> 2.70"
        }
    }
}
provider  aws  {
   profile = "default"
   region = "us-west-2"
}

resource "aws_instance" "example" {
     ami = "ami-830c94e3"
     instance_type = "t2.micro"
}
```

```
$ terraform show
# aws_instance.example:
resource "aws_instance" "example" {
    ami                          = "ami-830c94e3"
    arn                          = "arn:aws:ec2:us-east-1:130490850807:instance/i-
0bbf06244e44211d1"
    associate_public_ip_address  = true
    availability_zone            = "us-west-2"
    cpu_core_count               = 1
    cpu_threads_per_core         = 1
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-0bbf06244e44211d1"
    instance_state               = "running"
    instance_type                = "t2.micro"
    ipv6_address_count           = 0
    ipv6_addresses               = []
    monitoring                   = false
    primary_network_interface_id = "eni-0f1ce5bdae258b015"
    private_dns                  = "ip-172-31-61-141.ec2.internal"
    private_ip                   = "172.31.61.141"
    public_dns                   = "ec2-54-166-19-naws.com"
    public_ip                    = "54.166.19.244"
    security_groups              = [
        "default",
    ]
    source_dest_check            = true
    subnet_id                    = "subnet-1facdf35"
    tenancy                      = "default"
    volume_tags                  = {}
    vpc_security_group_ids       = [
        "sg-5255f429",
    ]
```

Ansible vs Terraform?

ChatGPT and DevOps?

Tampere University

Example

Kari: Can you provide me with terraform configuration file to create two small linux virtual machines connected with a private virtual network in AWS?

ChatGPT: Certainly! Below is an example Terraform configuration file (usually named main.tf) that you can use to create two small Linux virtual machines (EC2 instances) connected with a private virtual network (VPC) in AWS. This example assumes you want to use the Amazon Linux 2 AMI and a t2.micro instance type for simplicity.

✂️

Make sure to replace the placeholder values with your actual AWS region, availability zone, AMI ID, key pair name, etc. You can find the appropriate AMI ID for your region on the AWS console or using AWS CLI commands.

After creating the Terraform configuration file, you can run the following Terraform commands in the terminal:

```
terraform init
terraform apply
```

Kari: How about the same configuration for Azure?

ChatGPT: Certainly! Below is an example Terraform configuration file (usually named main.tf) to create two small Linux virtual machines connected with a private virtual network in Microsoft Azure. This example uses Ubuntu Server 20.04 LTS for the virtual machines.

✂️

Make sure to replace the placeholder values with your actual Azure region, admin username, admin password, etc.

After creating the Terraform configuration file, you can run the following Terraform commands in the terminal:

```
# Configure the AWS provider
provider "aws" {
  region = "your_aws_region" # Replace with
           your desired AWS region
}

# Create a Virtual Private Cloud (VPC)
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
}

# Create a subnet within the VPC
resource "aws_subnet" "my_subnet" {
  vpc_id                  =
aws_vpc.my_vpc.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       =
"your_preferred_availability_zone" # Replace
with your desired availability zone
}
```

```
# Configure the Azure provider
provider "azurerm" {
  features = {}
}
# Create a resource group
resource "azurerm_resource_group"
"my_resource_group" {
  name     = "myResourceGroup"
  location = "your_azure_region" # Replace with
             your desired Azure region
}

# Create a virtual network
resource "azurerm_virtual_network" "my_vnet" {
  name                = "myVNet"
  resource_group_name =
azurerm_resource_group.my_resource_group.name
  address_space       = ["10.0.0.0/16"]
}

# Create a subnet within the virtual network
resource "azurerm_subnet" "my_subnet" {
  name                 = "mySubnet"
  resource_group_name  =
azurerm_resource_group.my_resource_group.name
  virtual_network_name =
azurerm_virtual_network.my_vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}
```

```
# Create a security group for the instances
resource "aws_security_group"
"my_security_group" {
  name        = "my_security_group"
  description = "Allow SSH inbound traffic"

  vpc_id = aws_vpc.my_vpc.id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```
# Create a network security group
resource "azurerm_network_security_group"
"my_nsg" {
  name                = "myNSG"
  resource_group_name =
azurerm_resource_group.my_resource_group.name

  security_rule {
    name                       = "AllowSSH"
    priority                   = 1001
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}
```

```
}
```

```
# Create a network interface for each VM
resource "azurerm_network_interface" "my_nic_1"
{
  name                = "myNIC1"
  resource_group_name =
azurerm_resource_group.my_resource_group.name

  ip_configuration {
    name          = "myNIC1Config"
    subnet_id     = azurerm_subnet.my_subnet.id
    private_ip_address_allocation = "Dynamic"
  }
}

resource "azurerm_network_interface" "my_nic_2"
{
  name                = "myNIC2"
  resource_group_name =
azurerm_resource_group.my_resource_group.name

  ip_configuration {
    name          = "myNIC2Config"
    subnet_id     = azurerm_subnet.my_subnet.id
    private_ip_address_allocation = "Dynamic"
  }
}
```

```
# Create two EC2 instances
resource "aws_instance" "my_instance_1" {
  ami           = "ami-xxxxxxxxxxxxxxxxx" #
Replace with the Amazon Linux 2 AMI ID
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id
  key_name      = "your_key_pair_name" #
Replace with your EC2 key pair name
  security_group_ids =
[aws_security_group.my_security_group.id]
  tags = {
    Name = "my_instance_1"
  }
}

resource "aws_instance" "my_instance_2" {
  ami           = "ami-xxxxxxxxxxxxxxxxx" #
Replace with the Amazon Linux 2 AMI ID
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id
  key_name      = "your_key_pair_name" #
Replace with your EC2 key pair name
  security_group_ids =
[aws_security_group.my_security_group.id]

  tags = {
    Name = "my_instance_2"
  }
}
```

```
# Create two virtual machines
resource "azurerm_virtual_machine" "my_vm_1" {
  name                  = "myVM1"
  resource_group_name   =
azurerm_resource_group.my_resource_group.name
  location              =
azurerm_resource_group.my_resource_group.location
  availability_set_id   = null
  network_interface_ids =
[azurerm_network_interface.my_nic_1.id]
  vm_size = "Standard_B1s"
  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "20.04-LTS"
    version   = "latest"
  }
  os_profile {
    computer_name  = "myVM1"
    admin_username = "your_admin_username" #
Replace with your desired admin username
    admin_password = "your_admin_password"
  }
  os_profile_linux_config {
    disable_password_authentication = false
  }
}
resource "azurerm_virtual_machine" "my_vm_2" {
```

```
# Create two EC2 instances
resource "aws_instance" "my_instance_1" {
  ami           = "ami-xxxxxxxxxxxxxxxxx" #
Replace with the Amazon Linux 2 AMI ID
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id
  key_name      = "your_key_pair_name" #
Replace with your EC2 key pair name
  security_group_ids =
[aws_security_group.my_security_group.id]
  tags = {
    Name = "my_instance_1"
  }
}

resource "aws_instance" "my_instance_2" {
  ami           = "ami-xxxxxxxxxxxxxxxxx" #
Replace with the Amazon Linux 2 AMI ID
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.my_subnet.id
  key_name      = "your_key_pair_name" #
Replace with your EC2 key pair name
  security_group_ids =
[aws_security_group.my_security_group.id]

  tags = {
    Name = "my_instance_2"
  }
}
```

```
# Create two virtual machines
resource "azurerm_virtual_machine" "my_vm_1" {
  name                  = "myVM1"
  resource_group_name   =
azurerm_resource_group.my_resource_group.name
  location              =
azurerm_resource_group.my_resource_group.location
  availability_set_id   = null
  network_interface_ids =
[azurerm_network_interface.my_nic_1.id]
  vm_size = "Standard_B1s"
  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "20.04-LTS"
    version   = "latest"
  }
  os_profile {
    computer_name  = "myVM1"
    admin_username = "your_admin_username" #
Replace with your desired admin username
    admin_password = "your_admin_password"
  }
  os_profile_linux_config {
    disable_password_authentication = false
  }
}
resource "azurerm_virtual_machine" "my_vm_2" {
```

```
provider "aws" {
  region = "eu-west-1"
}

#############################################################
# Data sources to get VPC, subnet, security group and AMI details
#############################################################
data "aws_vpc" "default" {
  default = true
}

data "aws_subnet_ids" "all" {
  vpc_id = "${data.aws_vpc.default.id}"
}

data "aws_ami" "amazon_linux" {
  most_recent = true

  filter {
    name = "name"

    values = [
      "amzn-ami-hvm-*-x86_64-gp2",
    ]
  }
```

An example from Terraform

```
 filter {
    name = "owner-alias"

    values = [
      "amazon",
    ]
  }
}

module "security_group" {
  source = "terraform-aws-modules/security-group/aws"

  name        = "example"
  description = "Security group for example usage with EC2 instance"
  vpc_id      = "${data.aws_vpc.default.id}"

  ingress_cidr_blocks = ["0.0.0.0/0"]
  ingress_rules       = ["http-80-tcp", "all-icmp"]
  egress_rules        = ["all-all"]
}
```

```
resource "aws_eip" "this" {
  vpc      = true
  instance = "${module.ec2.id[0]}"
}

module "ec2" {
  source = "../../"

  name                        = "example"
  ami                         = "${data.aws_ami.amazon_linux.id}"
  instance_type               = "t2.micro"
  subnet_id                   = "${element(data.aws_subnet_ids.all.ids, 0)}"
  vpc_security_group_ids      = ["${module.security_group.this_security_group_id}"]
  associate_public_ip_address = true
}
```

ChatGPT and DevOps?

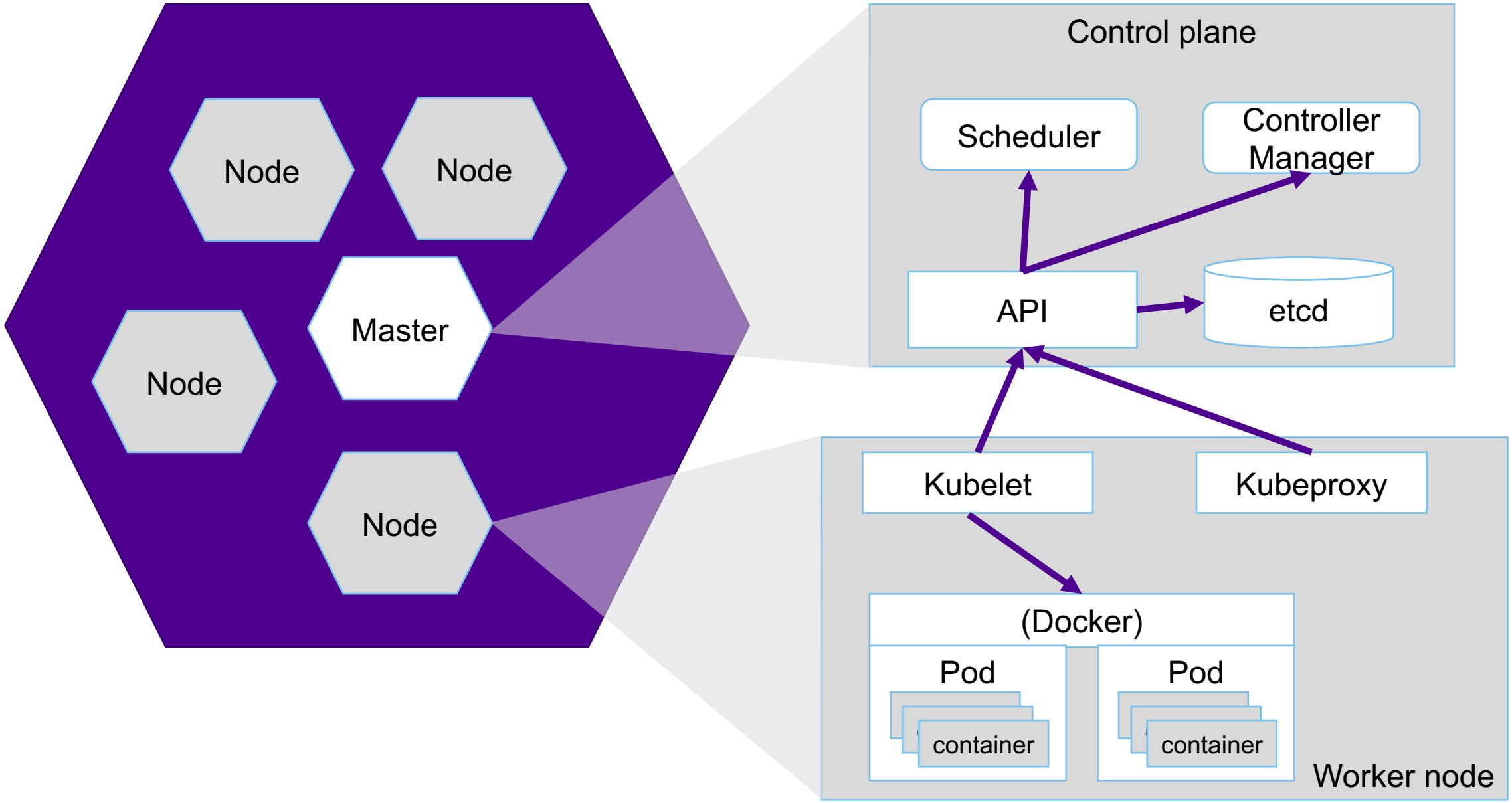
Cross-platform IaC

IaC tools

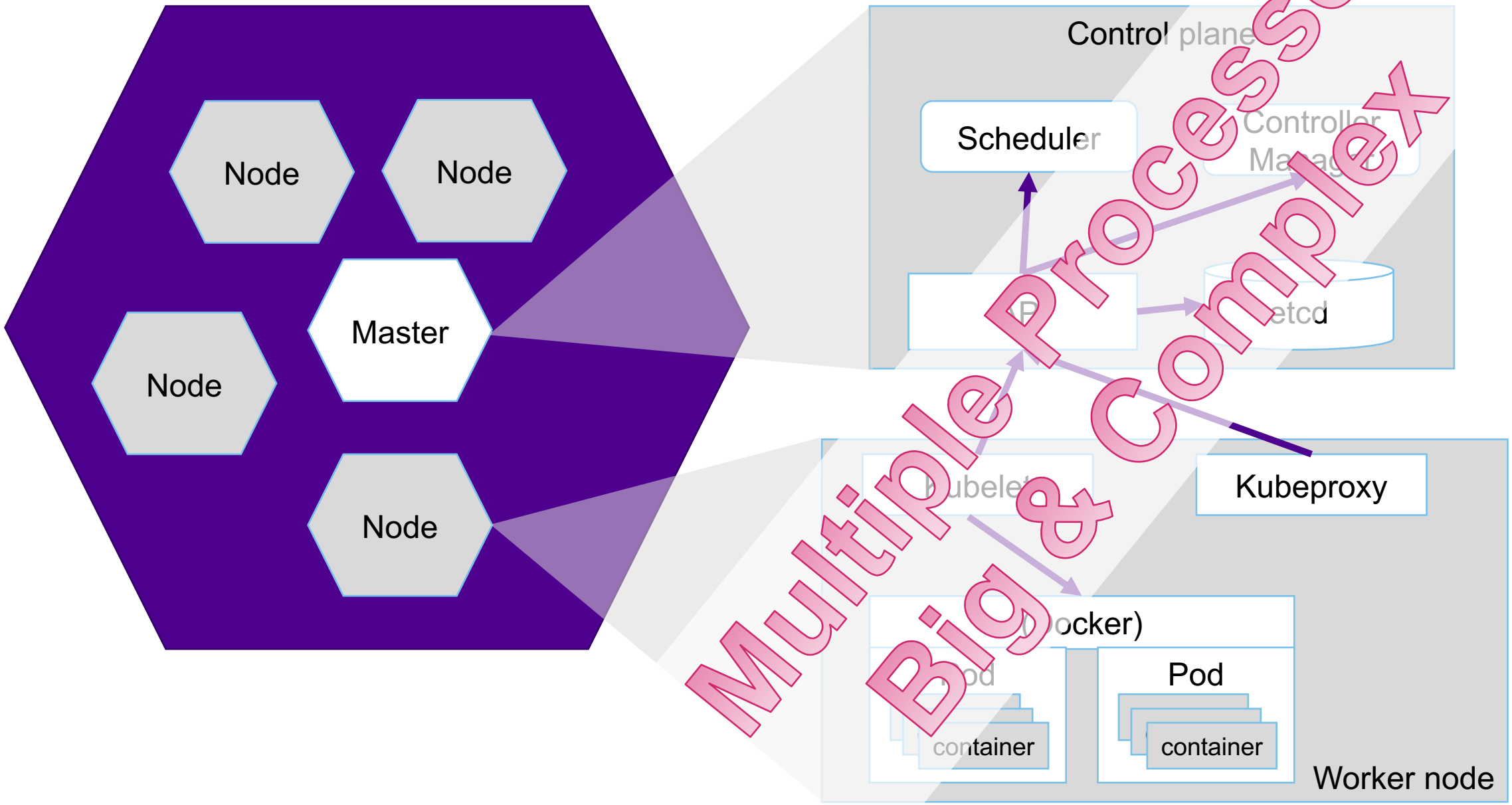
- The most common alternatives:
  - AWS CloudFormation
  - Microsoft Azure Resource Manager (ARM)
  - Terraform

- Other alternatives
  - Pulumi: programmable

Kubernetes Kluster



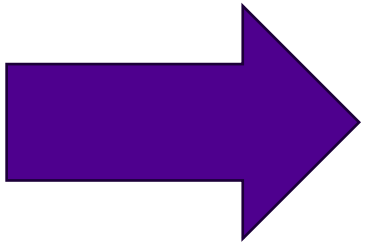
Kubernetes Kluster



K3S

Kubernetes

8

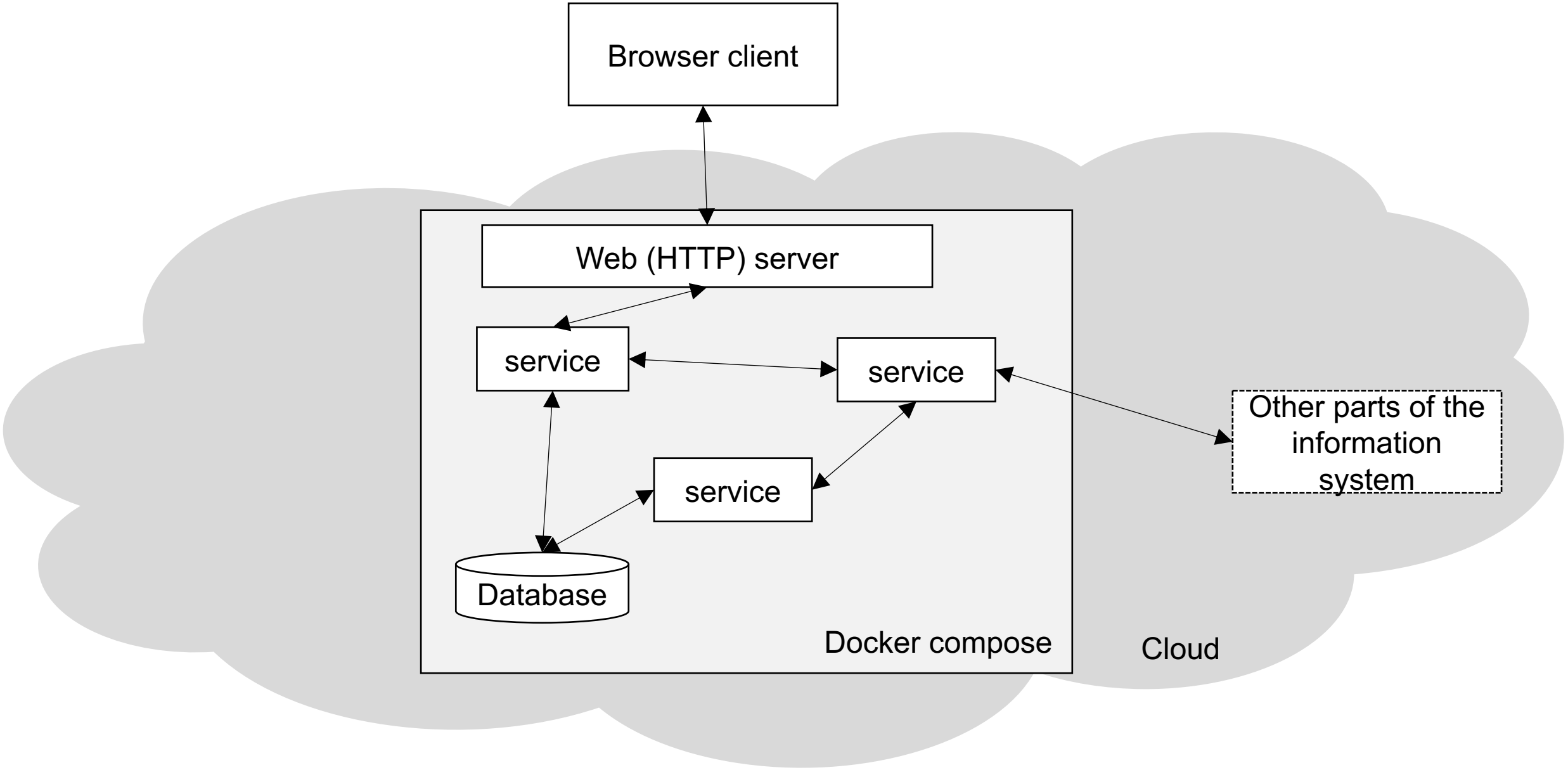


K3S

- single binary
- smaller
- simpler
- can be used in edge

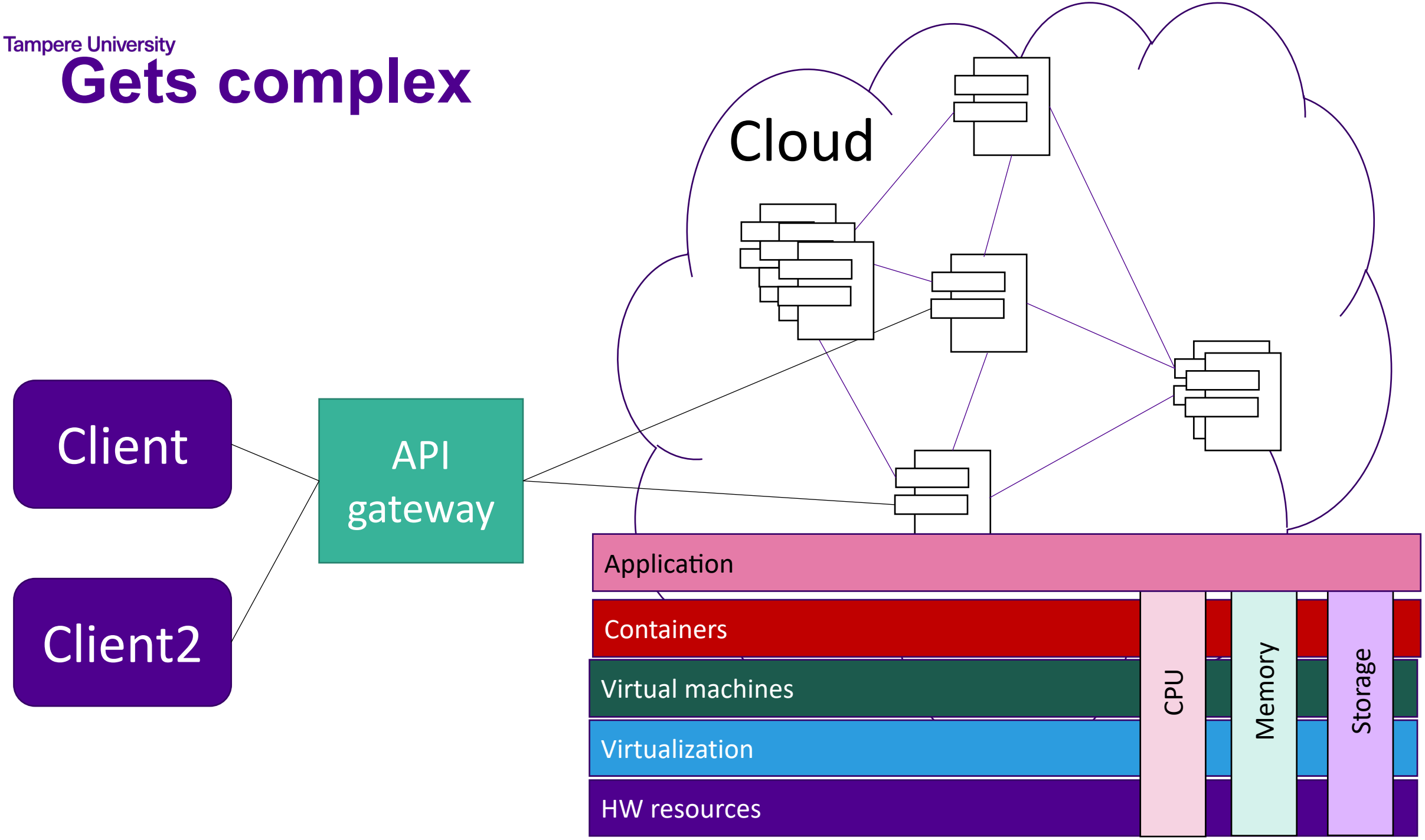
Tampere University

Dependency and version problems



Tampere University

Gets complex



Gets more complex