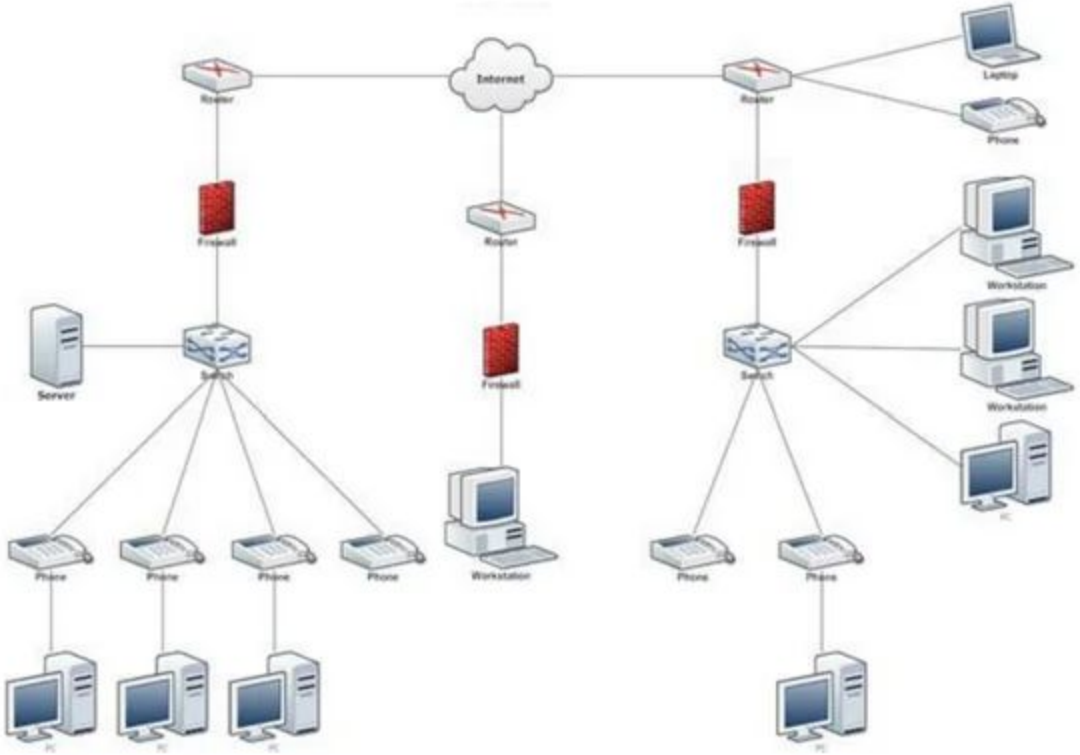
Introduction to Infrastructure-as-Code

CS 40 | January 31, 2024

Agenda

1. A Brief Historical Overview
2. Fundamental Principles of IaC
3. 5 Minute Break
4. Prominent IaC Tools

History


Internet
Router
Router
Router
Laptop
Phone
Firewall
Firewall
Firewall
Workstation
Server
Switch
Switch
Firewall
Workstation
Phone
Phone
Phone
Phone
Workstation
Phone
Phone

DELL Technologies

Search Dell

Sign In | Contact Us | US/EN | Cart

APEX | IT Infrastructure | Computers & Accessories | Services | Support | Deals

USA > Servers, Storage & Networking > Servers > Rack Servers > PowerEdge R750xa > PowerEdge R750XA Rack Server

Intel® Xeon® Scalable Processors
Learn More about Intel

PowerEdge R750XA Rack Server

Flagship server for GPU-based workloads

Improved air-cooling and expansion potential. Designed to accelerate performance AI/ML/DL [illegible] ute, performance graphics, and more.

Starting at $30,546.00
Financing Offers
Learn More

Add to Cart

Tech Specs & Customization | Features | Reviews | Drivers, Manuals & Support

Pow[illegible]R750XA Rack Server

Share

Selections may result in additional updates to the overall configuration, which may impact the price for Support and Services and the total overall price and savings for this product.

Components

Base

PowerEdge R750XA Server	**Selected**

Estimated Value	~~$50,910.00~~
Total Savings	$20,364.00
Shipping	Free
Dell Price	**$30,546.00**

Selections may result in additional updates to the overall configuration, which may impact the price for Support and Services and the total overall price and savings for this product.

```
opa eval -b autograder/rules/ -i <(jq -s 'reduce .[] as $item ({}; .Resources += $item.Resources) |
del(.Resources.CDKMetadata)' cdk/cdk.out/yoctogram-network-stack.template.json
cdk/cdk.out/yoctogram-data-stack.template.json cdk/cdk.out/yoctogram-compute-stack.template.json)
-f json 'data.rules.main' | jq -r .result[].expressions[].value.violations[]

jq -r '"\( .resources[] | .asn )^\(.site)"' < output.jsonl | sort | uniq | grep -v 'UNKNOWN^' |
grep -v 'null^' | grep -vf <(jq -r '"\(.asn)^\(.site)"' < output.jsonl) | datamash -t^ -g 1 count 2
| sort -k2 -t^ -n
```

```
deduplicate() {
    local displayflag="$1"
    shift
    local songlist_verbose=$($MPC -f "$displayflag" find "$@" | sed -E 's#(.*)\) @ (.*/)?(.*)#\1) @ \3 @ \2\3#' | awk -F ' @ ' '{printf "%s @ %s
    if [[ "$(wc -l <<< "$songlist_verbose")" -gt 1 ]]
    then
        # lots of songs, we need to disambiguate
        local songlist="$(awk -F ' @ ' 'BEGIN{OFS=" @ ";} {print $1,$2,$3}' <<< $songlist_verbose)"
        local chosen_song_entry="$(select_from "$songlist" "song (disambiguation)")"
        # if song empty (e.g. cancelled selection), break out of function
        [[ -z "$chosen_song_entry" ]] && return 1
        # not perfect: doesn't technically only match beginning of line
        grep -F "$chosen_song_entry" <<< "$songlist_verbose" | awk -F ' @ ' '{print $4}'
    else
        # 0 or 1, so just return the output
        awk -F ' @ ' '{print $4}' <<< "$songlist_verbose"
    fi
```

Some General Challenges:

- How do I keep track of what my infrastructure looks like?
  - A: A network diagram, (outdated) documentation, and a lot of accumulated knowledge that no one ever bothered to write down

- How can I add more infrastructure?
  - Buy more servers, then stick them on a rack and connect them to the internal network

- How can I configure new infrastructure?
  - Run a giant magical bash script that you better hope works
  - “Saligrama’s Law of Brittleness”: The brittleness of a shell script grows exponentially with its length

Fundamental Principles of IaC

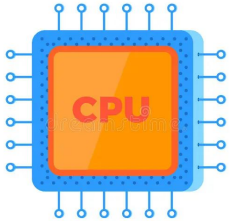
Goal: Manage infrastructure like you manage code

Code

- Written in a programming language

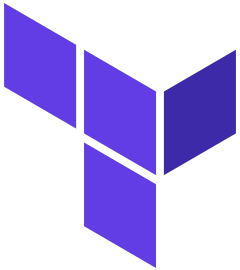
- Managed using *version control* (Git)
  - Keep track of changes

- Run on underlying hardware



Infrastructure as Code

- Written in a programming language
  - Terraform, CDK, etc
- Managed using *version control* (Git)
- Run on underlying hardware
  - Some infrastructure provider



Declarative vs Imperative Paradigm

- **Declarative**: declare the end state of infrastructure, don't specify steps to get there

- **Imperative**: write a series of steps on how to get to the end state

- Q: Should IaC be declarative or imperative?
  - A: Declarative – makes infrastructure simpler and easier to understand, avoids complexity
  - Use imperative methods only if your infrastructure is very complex and you need imperative features

State Management

- **State**: the resources that are currently deployed and their status

- IaC needs to manage state
  - Understand the difference between where your infrastructure currently is and where you want it to be
  - Also has the power to clean everything up if needed

- Workflow: *Plan*, then *apply*
  - Always check what changes your IaC is going to make!

Idempotency

- **Idempotency**: a property indicating that running the same deployment twice (or more times) has the effect as running it once
  - Concept from math: f(x) = f(f(x))

- Why is this a good thing?
  - Safety: prevent unnecessary resources from being deployed (saves money)
  - Simplicity: don't have to worry about current state of infrastructure if you run a deployment

5 Minute Break

Prominent IaC Tools

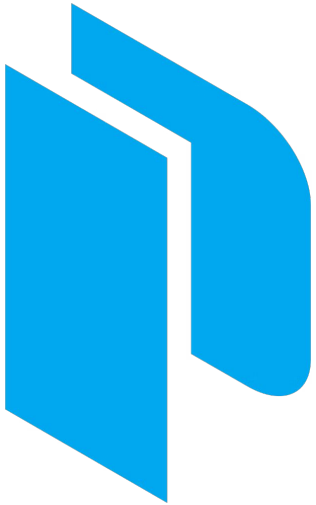
DANGER: Do not modify any resources created with IaC manually

Possible IaC Pipeline

1. Provision image
    - Packer, `docker build`
2. Deploy image
    - Terraform, AWS CDK, Pulumi, etc.
3. Runtime provisioning
    - `cloud-init`
4. Runtime management
    - Ansible, etc.

Packer

- Written in HCL or JSON
  - Hashicorp Configuration Language, essentially a less verbose version of JSON
- Two components:
  - **Builders**: spin up a live base image (e.g., an EC2 VM based on an Ubuntu AMI)
  - **Provisioners**: steps to configure the machine
    - Many types: shell, Ansible, file transfer, etc.
- Supports many different platforms
  - AWS, GCP, Azure, OpenStack, Proxmox
  - Can be extended using different providers

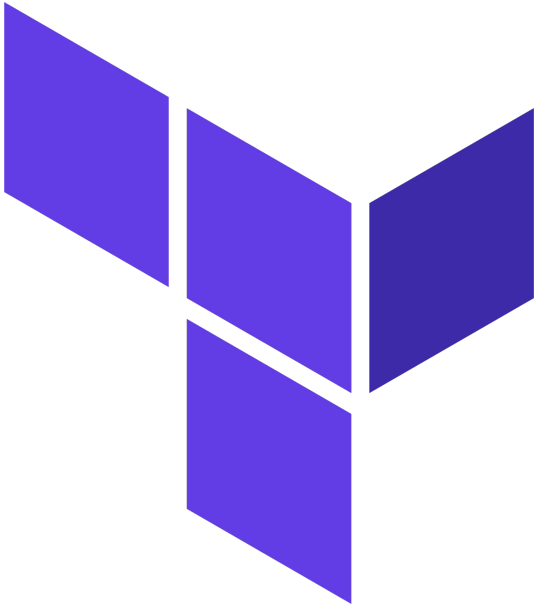


```
{
  "builders": [
    {
      "type": "amazon-ebs",
      "region": "us-west-2",
      "source_ami": "ami-076fa1ac9a95aef2e",
      "instance_type": "t4g.small",
      "ssh_username": "ubuntu",
      "ami_name": "cs40-assignment2-ubuntu-22.04-lts-with-tools-{{timestamp}}"
    }
  ],
  "provisioners": [
    {
      "type": "shell",
      "inline": ["/usr/bin/cloud-init status --wait"]
    },
    {
      "type": "file",
      "source": "99_user_defaults.cfg",
      "destination": "/tmp/99_user_defaults.cfg"
    },
    {
      "type": "shell",
      "inline": [ "sudo mv /tmp/99_user_defaults.cfg /etc/cloud/cloud.cfg.d/99_user_defaults.cfg" ]
    },
    {
      "type": "shell",
      "inline": [
        "sudo apt-get update",
        "sudo apt-get install -y jq",
        "sudo apt-get install -y ca-certificates curl gnupg unzip python3-venv python3-pip uidmap",
        "sudo mkdir -p /etc/apt/keyrings",
        "curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg",
        "echo 'deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node_20.x nodistro main' | sudo tee /etc/apt/sources.list.d/nodesource.list",
        "sudo apt-get update && sudo apt-get install -y nodejs",
        "sudo npm install -g aws-cdk",
        "rm -rf /tmp/aws"
      ]
    },
```

Demo: Packer

Terraform

- Written in HCL or JSON
- Purely declarative
- Contains a **provider** (AWS, Azure, GCP, Proxmox, etc) followed by a list of **resources**



```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "example" {
  ami           = "ami-0c94855ba95c574c8"
  instance_type = "t2.micro"

  tags = {
    Name = "example-instance"
  }
}
```

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_security_group" "example" {
  name        = "example-security-group"
  description = "Allow SSH and HTTP access"

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

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "example" {
  ami           = "ami-0c94855ba95c574c8"
  instance_type = "t2.micro"
  security_groups = [aws_security_group.example.id]

  tags = {
    Name = "example-instance"
  }
}
```

Terraform Licensing Drama

- Hashicorp develops Terraform (and many other cloud tools)
- Terraform licensed as *Mozilla Public License v2* until August 2023
- In August 2023, license changed to *Business Source License v2* (not open source) without warning
  - Widely regarded as a bad move by the IAC/DevOps community
- Open source alternative: OpenTofu
  - Replicates the Terraform binary functionality, but not Hashicorp's cloud services

Cloud Development Kits

- Examples: AWS CDK, CDKTF (Terraform), Pulumi

- Has the power to mix declarative and imperative programming paradigms
  - Possible footgun

- Written in a high level language
  - Python, Typescript, Go are common

AWS CDK Components

- **App**: The overall deployment, contains multiple stacks

- **Stack**: Basic unit of deployment
  - Can be deployed independently (assuming no dependencies)
  - Can have dependencies on other stacks
  - Should encapsulate individual components of deployment
    - Networking, compute, storage, etc

- **Environment**: AWS account and region where the deployment is to take place

```
props = Props()
env = cdk.Environment(account=settings.CDK_DEFAULT_ACCOUNT, region=settings.REGION)

dns_stack = DnsStack(app, f"{settings.PROJECT_NAME}-dns-stack", env=env)
props.network_hosted_zone = dns_stack.hosted_zone

network_stack = NetworkStack(
    app, f"{settings.PROJECT_NAME}-network-stack", props, env=env
)
props.network_vpc = network_stack.vpc
props.network_backend_certificate = network_stack.backend_certificate
props.network_frontend_certificate = network_stack.frontend_certificate

data_stack = DataStack(app, f"{settings.PROJECT_NAME}-data-stack", props, env=env)
props.data_aurora_db = data_stack.aurora_db
props.data_s3_public_images = data_stack.s3_public_images
props.data_s3_private_images = data_stack.s3_private_images
props.data_cloudfront_public_images = data_stack.cloudfront_public_images
props.data_cloudfront_private_images = data_stack.cloudfront_private_images

compute_stack = ComputeStack(
    app, f"{settings.PROJECT_NAME}-compute-stack", props, env=env
)

data_stack.add_dependency(network_stack)
compute_stack.add_dependency(data_stack)

app.synth()
```

Provisioning Resources in AWS CDK

- Each stack has a *constructor*, which creates all required resources for the stack

```python
class DnsStack(Stack):
    hosted_zone: r53.IHostedZone

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        self.hosted_zone = r53.HostedZone(
            self,
            f"{settings.PROJECT_NAME}-hosted-zone",
            zone_name=settings.SUNET_DNS_ROOT,
        )
```

- The call to `r53.HostedZone` creates the resource; the assignment to `self.hosted_zone` is only to communicate with other stacks

```python
from aws_cdk import (
    Stack,
    aws_ec2 as ec2,
    aws_route53 as r53
)

class ExampleStack(Stack):
    def __init__(
        self, scope: Construct, construct_id: str, **kwargs
    ) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # Import the existing Hosted Zone created earlier.
        hosted_zone = r53.HostedZone.from_lookup(
            self, "EXAMPLE_ZONE_ID", domain_name="example.infracourse.cloud"
        )

        instance = ec2.Instance(
            self,
            "example-ec2-instance",
            instance_type=ec2.InstanceType("t4g.small"),
            machine_image=ec2.MachineImage.latest_amazon_linux(
                cpu_type=ec2.AmazonLinuxCpuType.ARM_64
            ),
            vpc=ec2.Vpc.from_lookup(self, "VPC", is_default=True)
        )

        # Create a DNS record `a2-example.example.infracourse.cloud`
        # pointing at the EC2 instance's public IP address.
        dns_record = r53.ARecord(
            self,
            zone=hosted_zone,
            record_name="a2-example",
            target=r53.RecordTarget.from_ip_addresses(
                instance.instance_public_ip
            )
        )
```

AWS CDK Execution

- `cdk synth`: Synthesize the CDK to AWS CloudFormation
  - AWS CloudFormation: Amazon's proprietary IaC tool that allows an entire deployment to be specified as a single JSON file

- `cdk bootstrap`: Create IAM roles needed to deploy and S3 bucket to store deployment artifacts

- `cdk deploy`: Deploy the generated CloudFormation to your account
  - CloudFormation calls the AWS SDK to provision AWS resources

Demo: AWS CDK

Dangers of CDK

- Don't use control flow, loops, if statements, etc
  - Complicates your deployment
  - Makes it less clear
  - Can cause unintended behavior

- Don't get lost in all the types and features
  - *Type annotations are your friend!*

- KISS still applies

Criticisms of AWS CDK

- Leaky abstraction over Cloudformation
  - Cyclical dependencies → broken deployment with no obvious checks
  - Order of deployment isn't always the order the code is written in
- Opaque: Cloudformation is proprietary AWS code that doesn't run locally
- High level languages are a footgun
- General annoyances: slowness, resource limits

`cloud-init`

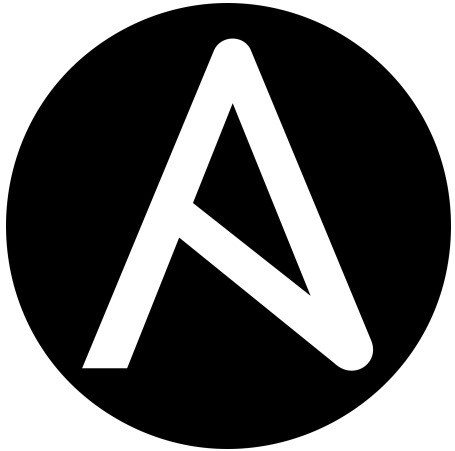
- Distribution and provider agnostic way of provisioning VMs and containers on first deploy
  - e.g. how AWS inserts your keypair into new EC2 vms

- Allows configuration using YAML files

- *Warning*: Make sure cloud-init is done before you do anything else on a machine

Ansible

- Written in YAML
- Allows you to connect to many VMs or containers over a protocol and then run tasks on all simultaneously
- Managed by an inventory file



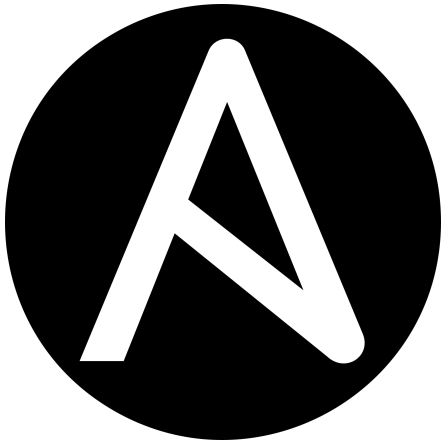
```
[linux]
192.168.170.32
192.168.170.132
192.168.170.216
192.168.170.222
192.168.171.9
192.168.171.12
192.168.171.236
192.168.172.93
192.168.172.204
192.168.172.223

[linux:vars]
ansible_user=administrator
```

```yaml
- name: CS40 demo
  hosts: linux
  become: yes
  tasks:
    - name: run a command
      shell: "hostname"
      register: output

    - name: print output
      debug:
        msg: "{{ output.stdout_lines[0] }}"

    - name: add a user
      user:
        name: test256
        state: present
```

Demo: Ansible

Ansible Fork Drama

- Ansible is developed by Red Hat
- In 2018, Red Hat is acquired by IBM



- Red Hat strips out many Ansible features and creates "Ansible Core"
- Community forks Ansible, still called "Ansible"
    - Just use the community version and don't think too hard

Next Lecture: Identity & Access Management (2/5)