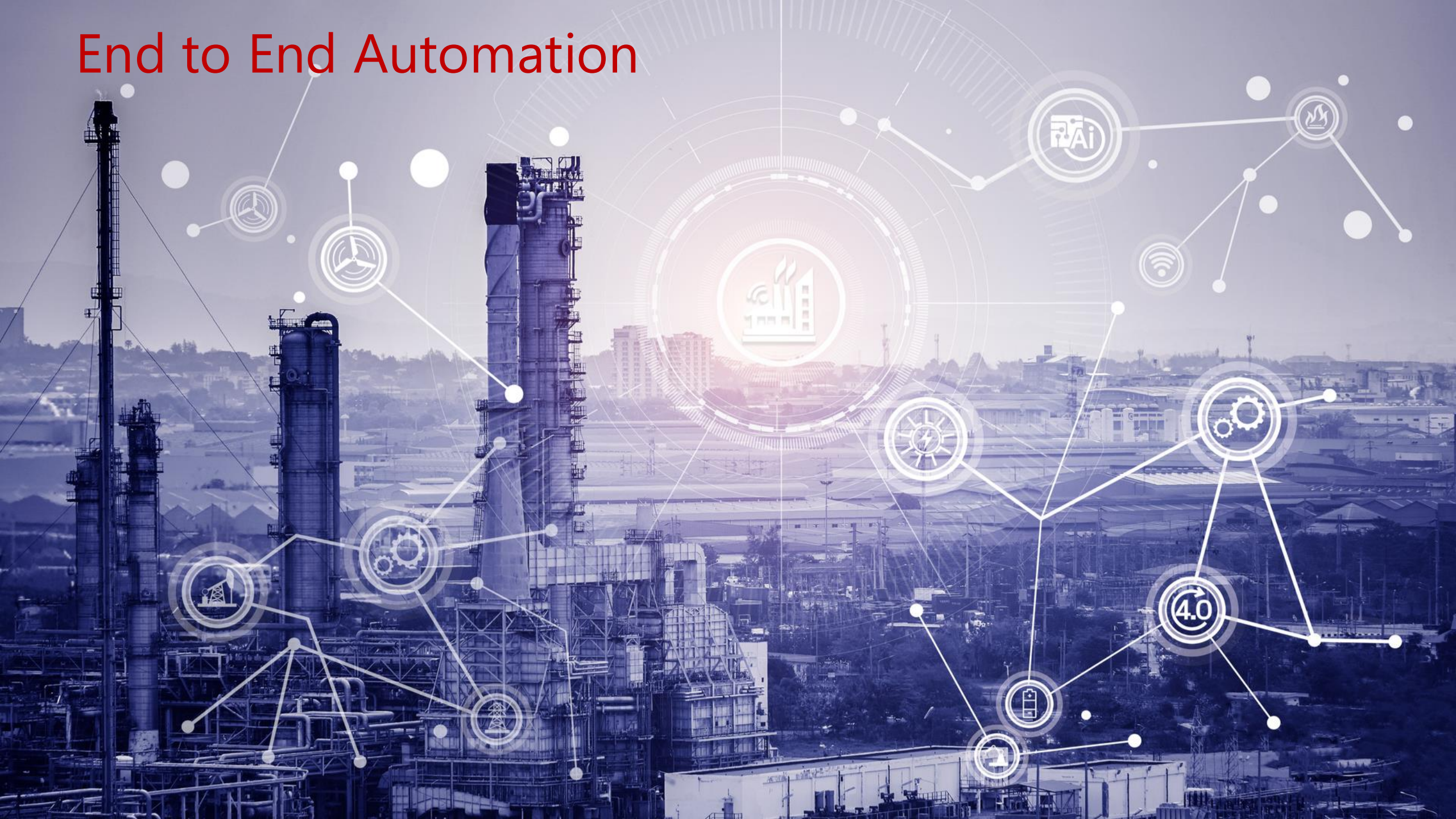
End to End Automation



TEKsystems Global Services

WORKFORCE DEVELOPMENT

Content Usage Parameters

Content refers to material including instructor guides, student guides, lab guides, lab or hands-on activities, computer programs, etc. designed for use in a training program

1

Content is subject to copyright protection

2

Content may only be leveraged by students enrolled in the training program

3

Students agree not to reproduce, make derivative works of, distribute, publicly perform and publicly display in any form or medium outside of the training program

4

Content is intended as reference material only to supplement the instructor-led training

Logistics

- Class Hours:
- Instructor will provide class start and end times.
- Breaks throughout class

- Telecommunication:
- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students

- Lunch:
- 1 hour 15 minutes

- Miscellaneous
- Courseware
- Bathroom

Course objectives

- Explain how DevSecOps improves the development and deployment of applications
- Understand Infrastructure as Code (IaC)
- Deploy infrastructure with Terraform
- Create and use Terraform modules
- Create and interact with Kubernetes
- Build microservices and deploy microservices applications
- Utilize Argo CD for GitOps

Hi!

Jason Smith

Cloud Consultant with a Linux sysadmin background.
Focused on cloud-native technologies: automation, containers & orchestration

<u>Expertise</u>

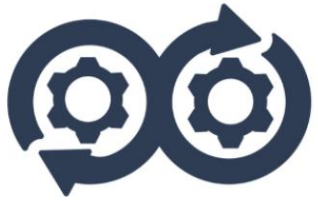
- Cloud
- Automation
- CICD
- Docker
- Kubernetes

Introduce Yourself

Time to introduce yourself:

- Name you prefer
- Your professional background
- Current responsibilities
- Familiarity with:
  - Git/Containers/Kubernetes
  - Terraform/Ansible
  - Argo CD
- Expectations and goals for class

What is DevSecOps?



DevSecOps

How to incorporate security within agile and DevOps practices?

DevOps vs DevSecOps

DevOps:
The goal of DevOps is to bring together formerly siloed roles, such as IT operations, development, and QA, to coordinate and produce better, more reliable products.

DevSecOps:
All stakeholders who participate in the application development lifecycle are responsible for the security of the final product.

DevOps: Fewer outages
DevSecOps: No data loss.

DevSecOps

The purpose and intent of DevSecOps is to build on the mindset that "everyone is responsible for security" with the goal of safely distributing security decisions at speed and scale to those who hold the highest level of context without sacrificing the safety required.

POP QUIZ: Security in SDLC

Does security hinder innovation?

DevSecOps motto

Software, Safer, Sooner

POP QUIZ: DISCUSSION

What is the primary difference in outcomes between DevOps and DevSecOps?

A) DevOps focuses on no data loss, while DevSecOps aims for fewer outages

B) DevOps brings siloed roles together for reliable products, while DevSecOps makes all stakeholders responsible for security

C) DevOps emphasizes security decisions, while DevSecOps prioritizes product reliability

D) DevOps requires quarterly scans, while DevSecOps uses automated pipelines

POP QUIZ: DISCUSSION

What is the primary difference in outcomes between DevOps and DevSecOps?

A) DevOps focuses on no data loss, while DevSecOps aims for fewer outages

B) DevOps brings siloed roles together for reliable products, while DevSecOps makes all stakeholders responsible for security

C) DevOps emphasizes security decisions, while DevSecOps prioritizes product reliability

D) DevOps requires quarterly scans, while DevSecOps uses automated pipelines

DevSecOps manifesto

Leaning in over Always Saying "No"
Data & Security Science over Fear, Uncertainty and Doubt
Open Contribution & Collaboration over Security-Only Requirements
Consumable Security Services with APIs over Mandated Security Controls & Paperwork
Business Driven Security Scores over Rubber Stamp Security
Red & Blue Team Exploit Testing over Relying on Scans and Theoretical Vulnerabilities
24x7 Proactive Security Monitoring over Reacting after being Informed of an Incident
Shared Threat Intelligence over Keeping Info to Ourselves
Compliance Operations over Clipboards & Checklists

POP QUIZ: DISCUSSION

What is the primary goal of DevSecOps?

A) To centralize security responsibilities with a dedicated team

B) To build on the mindset that everyone is responsible for security by safely distributing security decisions at speed and scale to those with the highest context without sacrificing safety

C) To prioritize innovation over security in application development

D) To eliminate the need for DevOps practices

POP QUIZ: DISCUSSION

What is the primary goal of DevSecOps?

A) To centralize security responsibilities with a dedicated team

B) To build on the mindset that everyone is responsible for security by safely distributing security decisions at speed and scale to those with the highest context without sacrificing safety

C) To prioritize innovation over security in application development

D) To eliminate the need for DevOps practices

POP QUIZ: DISCUSSION

According to the DevSecOps manifesto, what should replace "Fear, Uncertainty, and Doubt" in security practices?

A) Mandated Security Controls & Paperwork
B) Relying on Scans and Theoretical Vulnerabilities
C) Data and Security Science

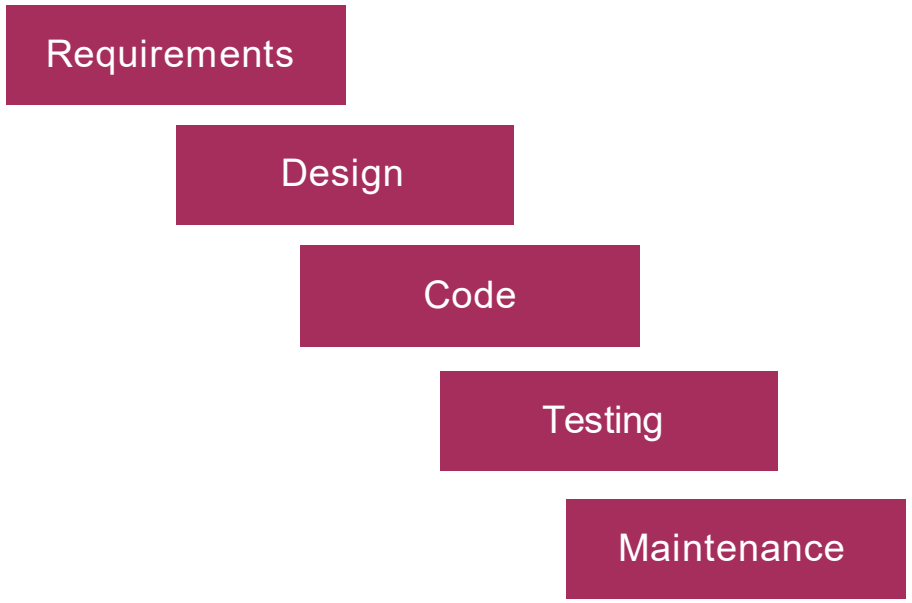
D) Keeping Info to Ourselves

POP QUIZ: DISCUSSION

According to the DevSecOps manifesto, what should replace "Fear, Uncertainty, and Doubt" in security practices?

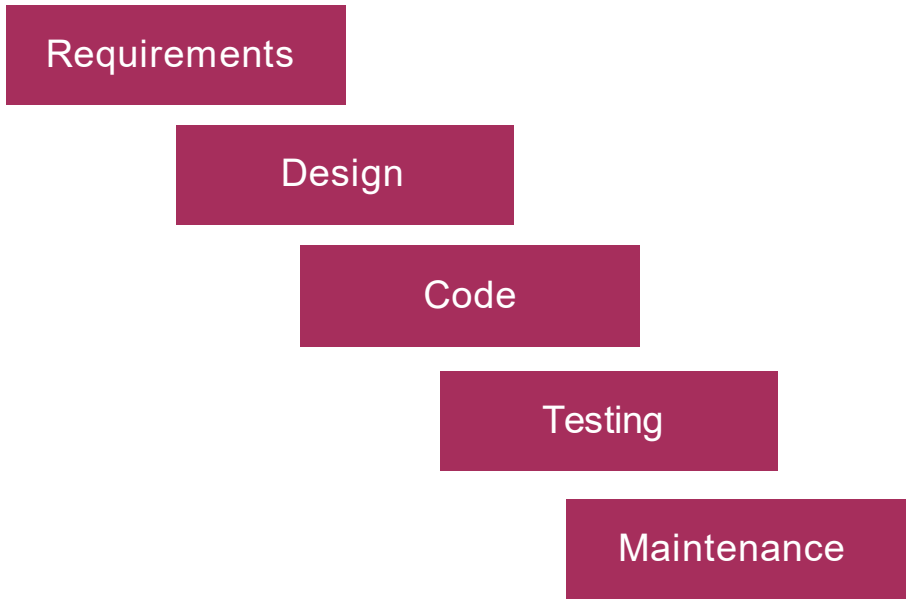
A) Mandated Security Controls & Paperwork
B) Relying on Scans and Theoretical Vulnerabilities
C) Data and Security Science
D) Keeping Info to Ourselves

Software Development Evolution

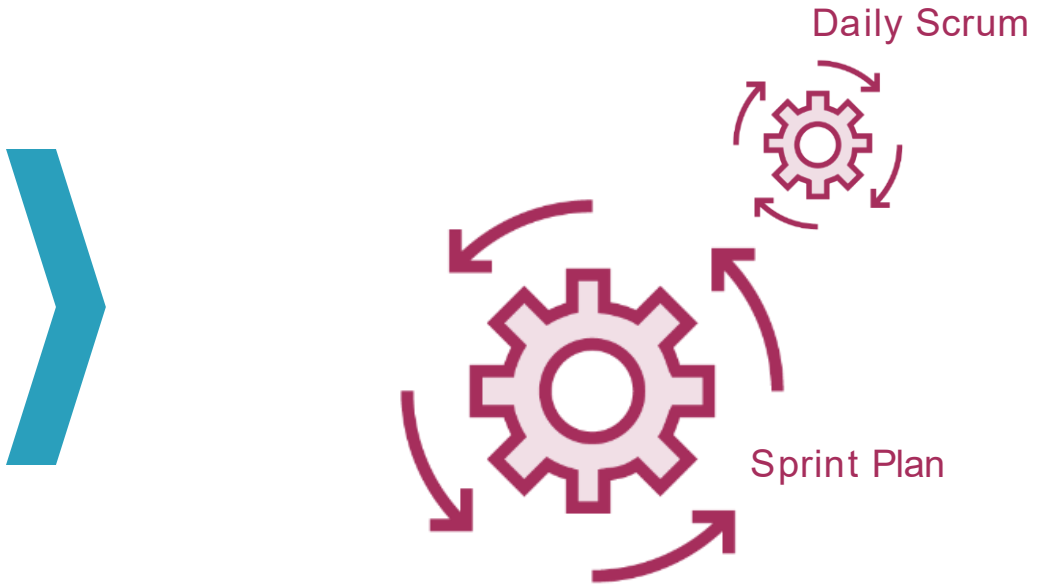


Waterfall Methodology

Software Development Evolution

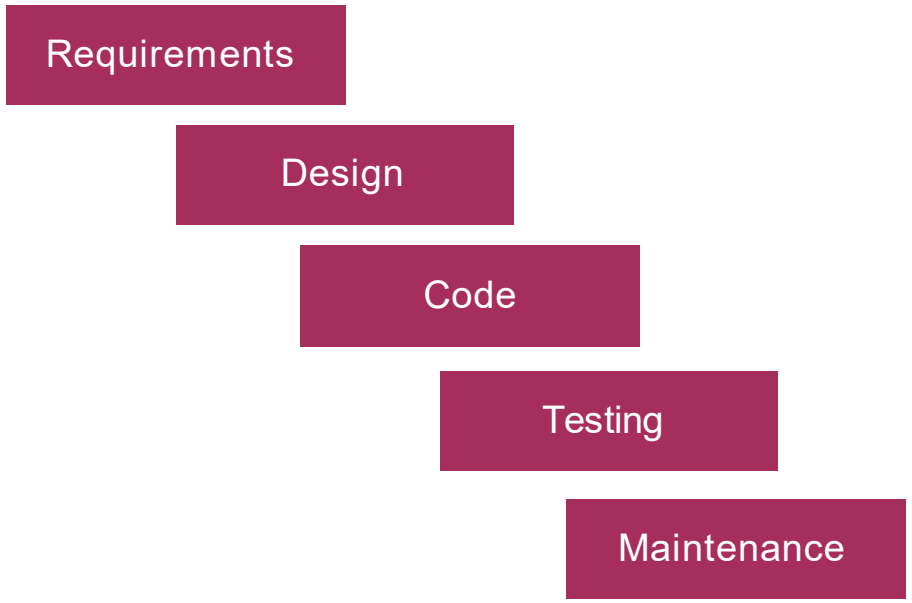


Waterfall Methodology

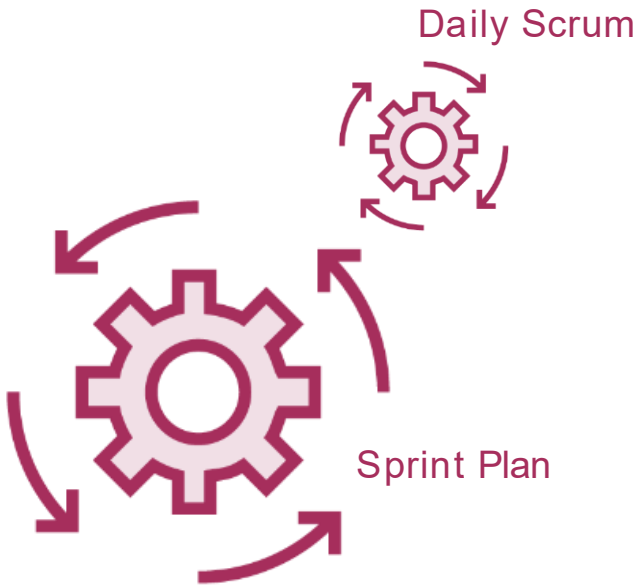


Agile Methodology

Software Development Evolution

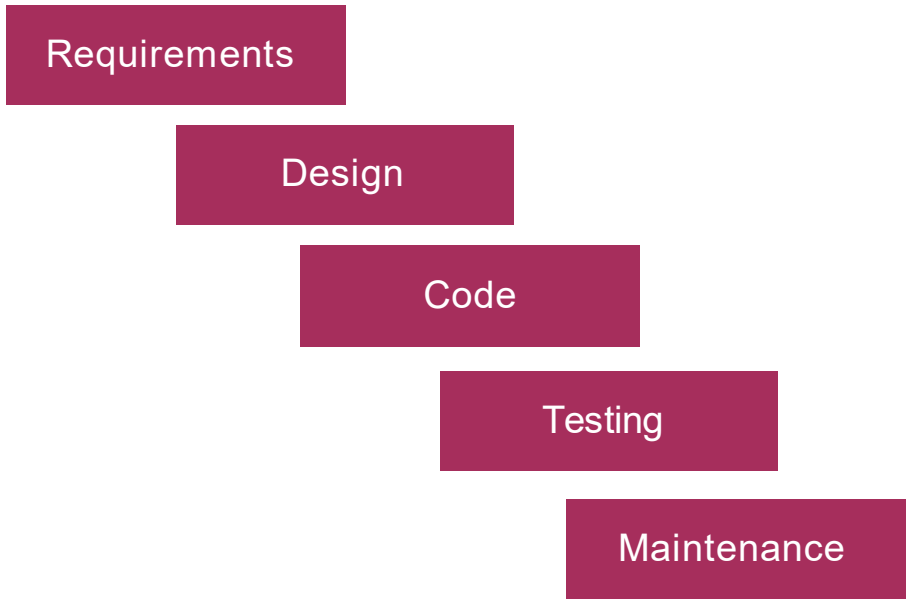


Waterfall Methodology

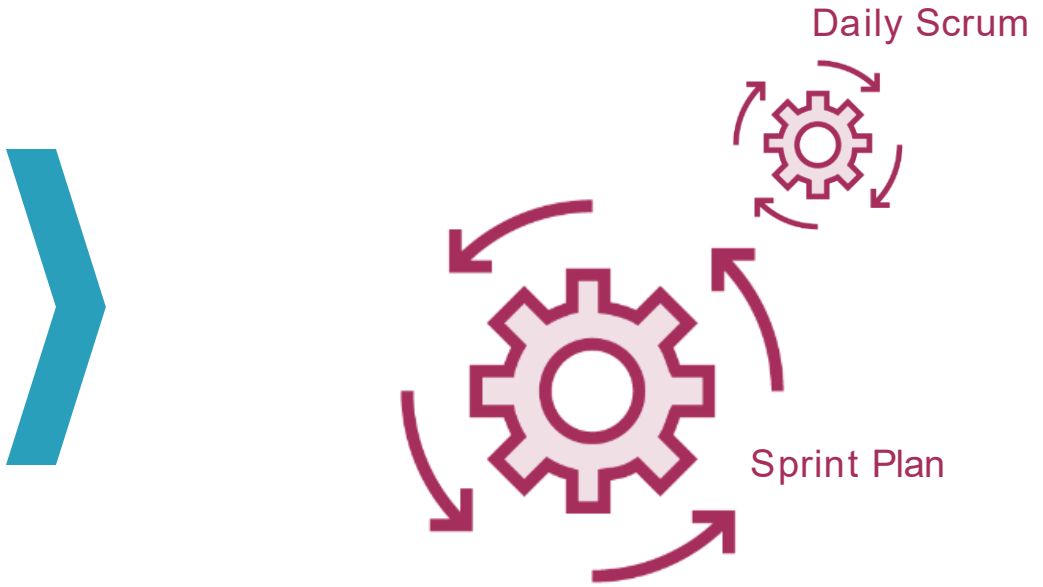


Agile Methodology

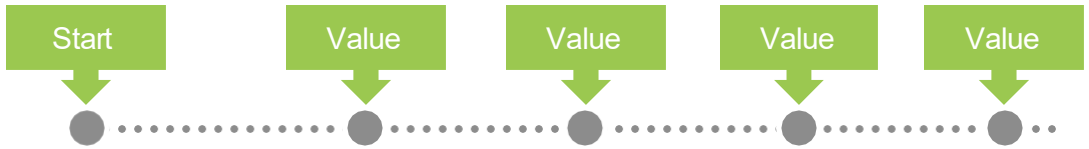
Software Development Evolution



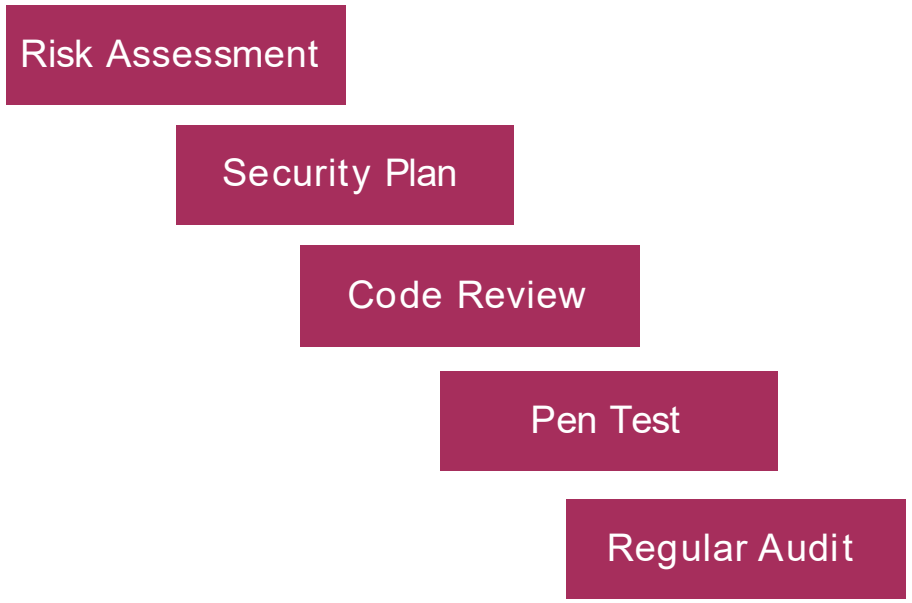
Waterfall Methodology



Agile Methodology

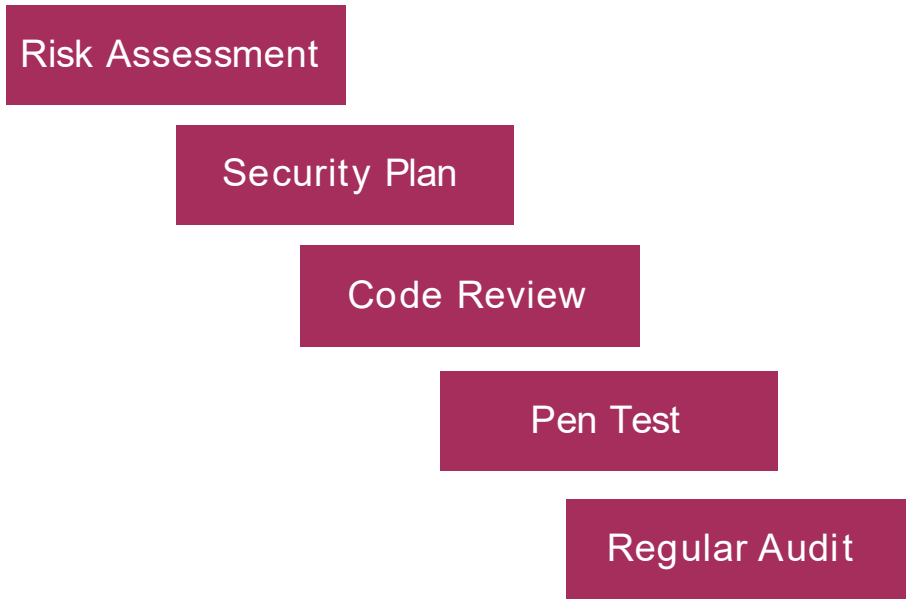


Information Security Friction



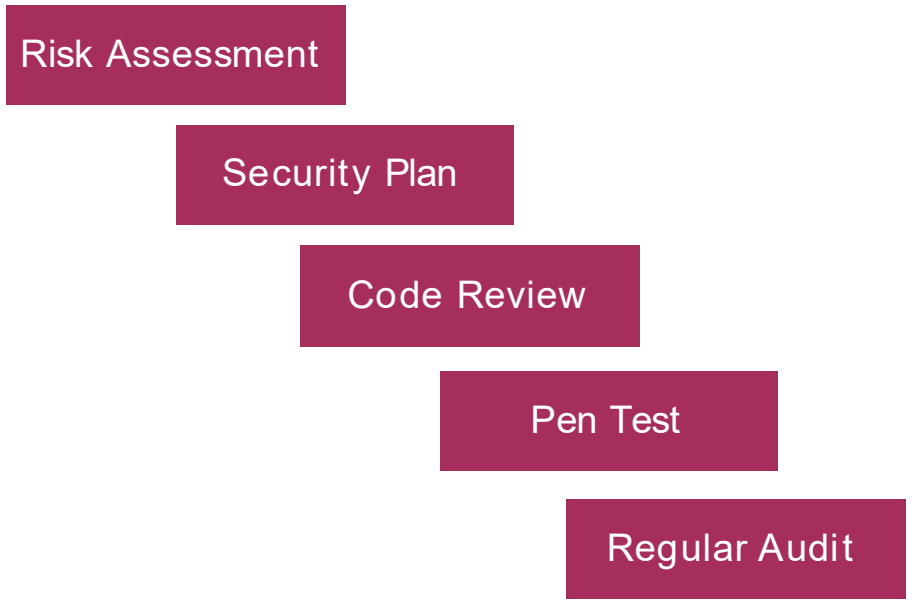
“Waterfall Methodology”

Information Security Friction



"Waterfall Methodology"

Information Security Friction



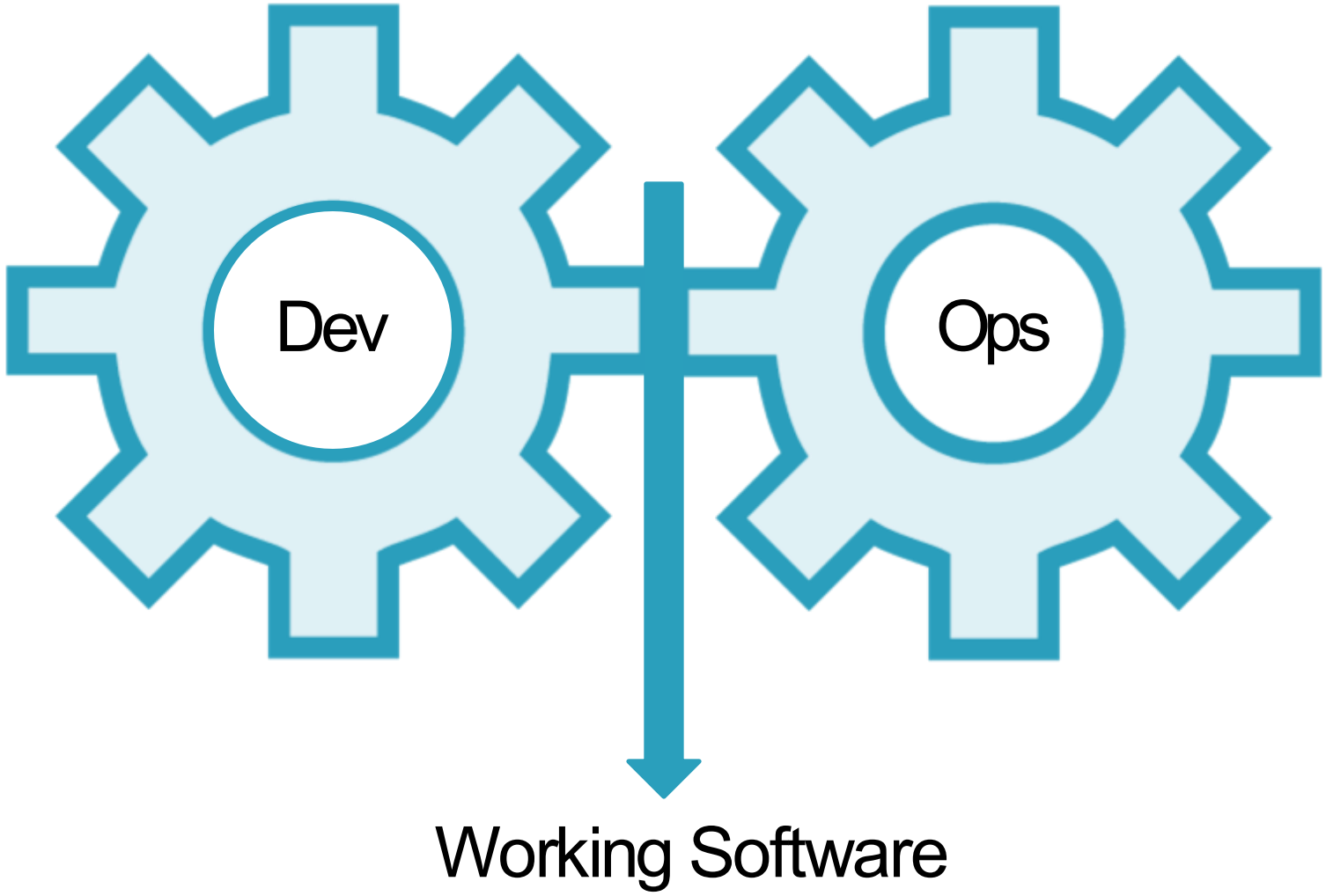
"Waterfall Methodology"

- Security as an after-thought
- Security "sign off" delays project
- Issues identified late in project
- Once-off point in time assessment
- Cost of re-testing is very high
- Security is too slow
- Not enough skills available to be secure
- Ratio of Security Experts to Dev Experts is very low

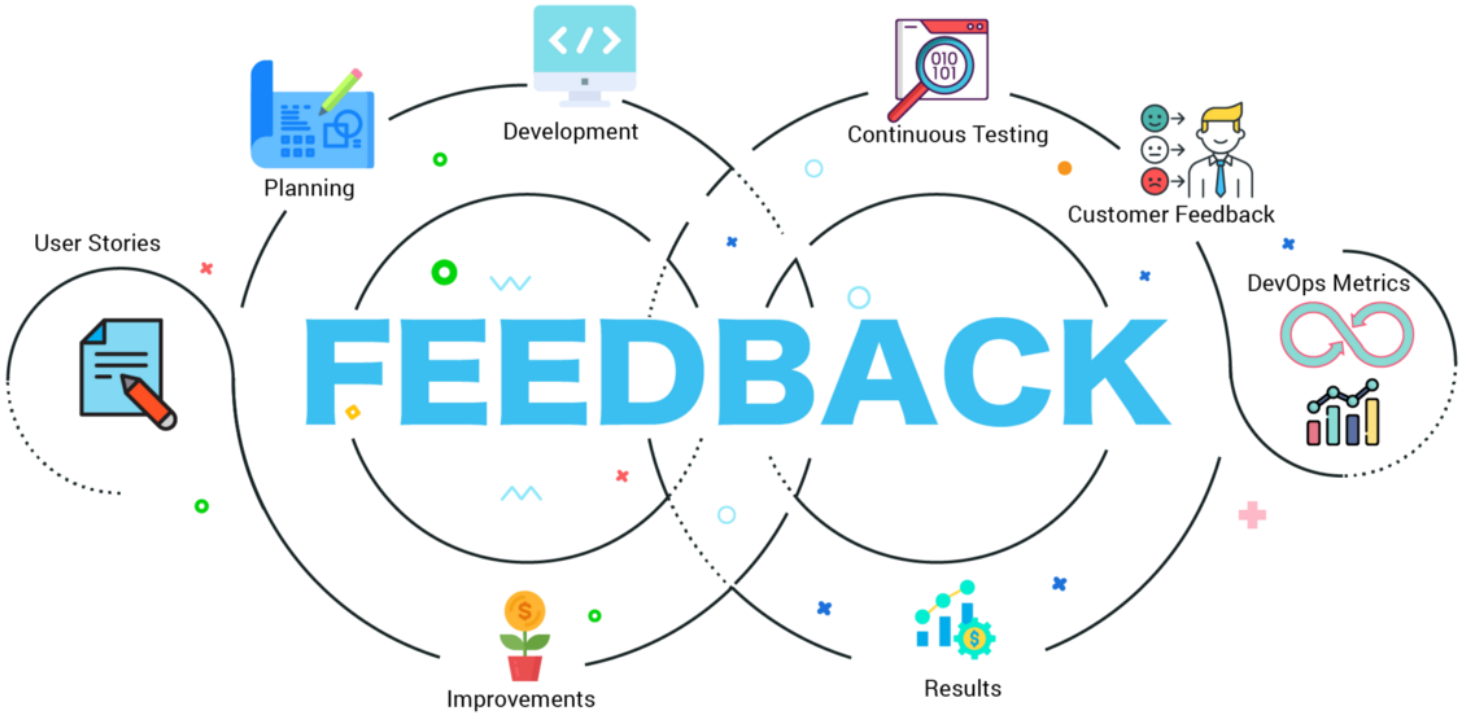
POP QUIZ: Security

Does security delay releases for your team?

DevOps



DevOps Feedback Loop



Measuring the impact of the development to know its effectiveness and continue to learn.

Real-world "Best Practice" example

PCI DSS V3.2.1

11.2 Run internal and external network vulnerability scans at least quarterly and after any significant change in the network.

Real-world "Best Practice" example

PCI DSS V3.2.1

11.2 Run internal and external network vulnerability scans at least quarterly and after any significant change in the network.

Real-world "Best Practice" example

PCI DSS V3.2.1

11.2 Run internal and external network vulnerability scans at least quarterly and after any significant change in the network.

Initiate a Vulnerability Scan once per quarter; have a human review the results and re-test to ensure "High Risk" vulnerabilities were addressed.

Real-world "Best Practice" example

PCI DSS V3.2.1

11.2 Run internal and external network vulnerability scans at least quarterly and after any significant change in the network.

Initiate a Vulnerability Scan once per quarter; have a human review the results and re-test to ensure "High Risk" vulnerabilities were addressed.

Undertake vulnerability scan as part of your software release pipeline, and do not release if "High Risk" vulnerabilities are identified.

POP QUIZ: DISCUSSION

In the evolution of software development, what issue is highlighted with security in the Waterfall Methodology?

A) Security is integrated early, reducing re-testing costs
B) Issues are identified late, making re-testing very expensive
C) It allows for daily scrums to address security friction

D) Security sign-offs speed up the project timeline

POP QUIZ: DISCUSSION

In the evolution of software development, what issue is highlighted with security in the Waterfall Methodology?

A) Security is integrated early, reducing re-testing costs

B) Issues are identified late, making re-testing very expensive

C) It allows for daily scrums to address security friction

D) Security sign-offs speed up the project timeline

POP QUIZ: DISCUSSION

What does the PCI DSS example illustrate as a "Best Practice" shift in DevSecOps?

A) Performing manual vulnerability scans quarterly with human review

B) Undertaking vulnerability scans as part of the software release pipeline and blocking releases with high-risk issues

C) Ignoring significant network changes for faster deployments

D) Relying on external teams for all compliance validations

POP QUIZ: DISCUSSION

What does the PCI DSS example illustrate as a "Best Practice" shift in DevSecOps?

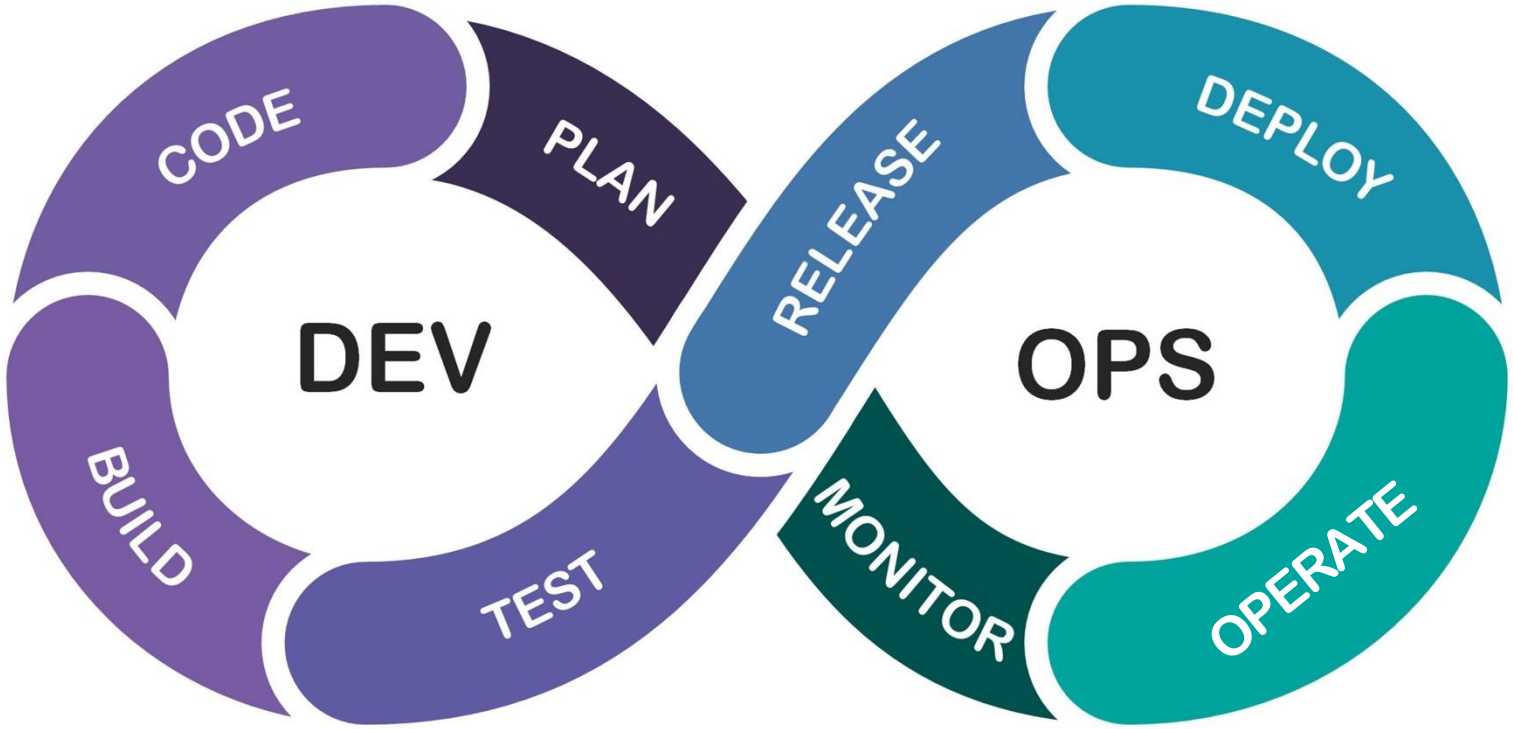
A) Performing manual vulnerability scans quarterly with human review

B) **Undertaking vulnerability scans as part of the software release pipeline and blocking releases with high-risk issues**

C) Ignoring significant network changes for faster deployments

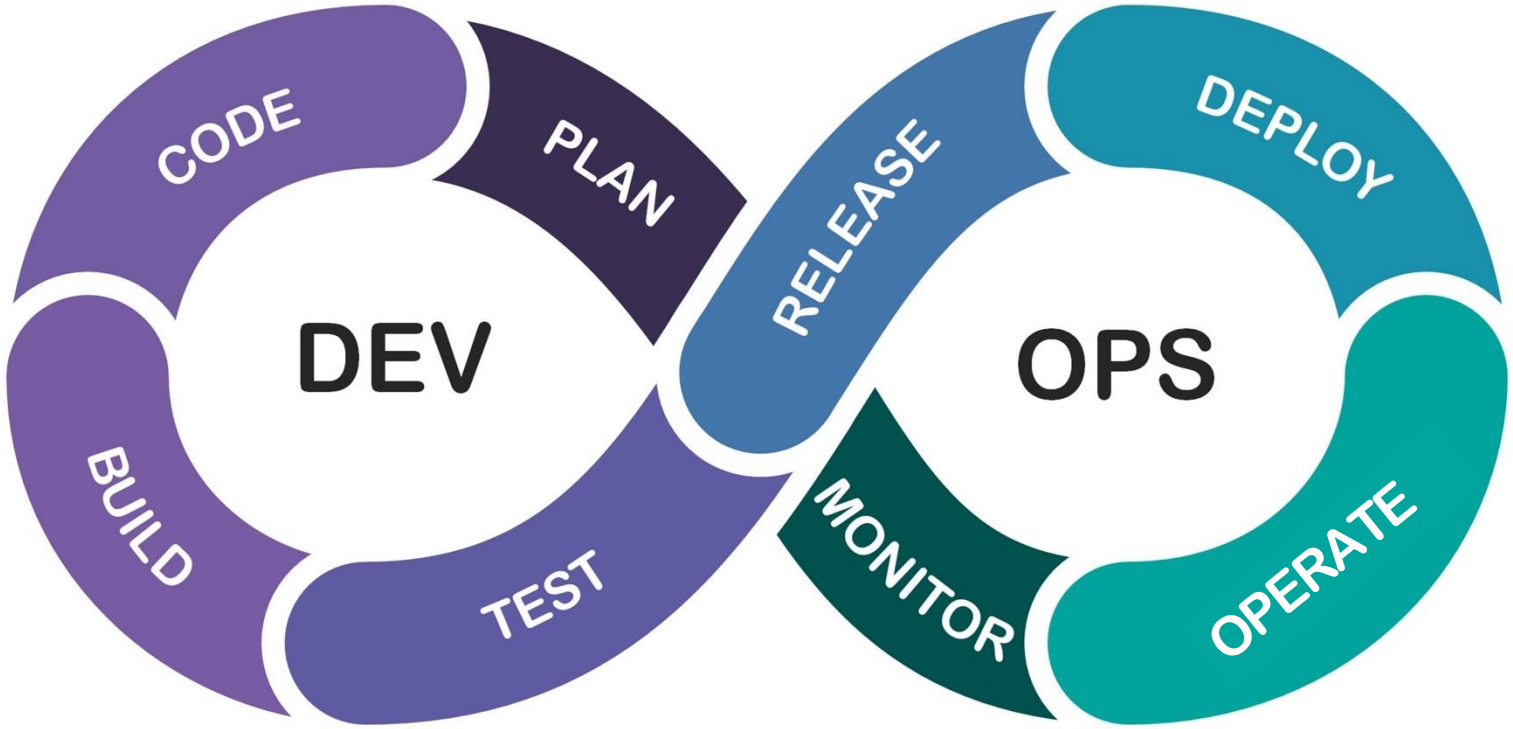
D) Relying on external teams for all compliance validations

Merge DevOps and Sec



Merge DevOps and Sec

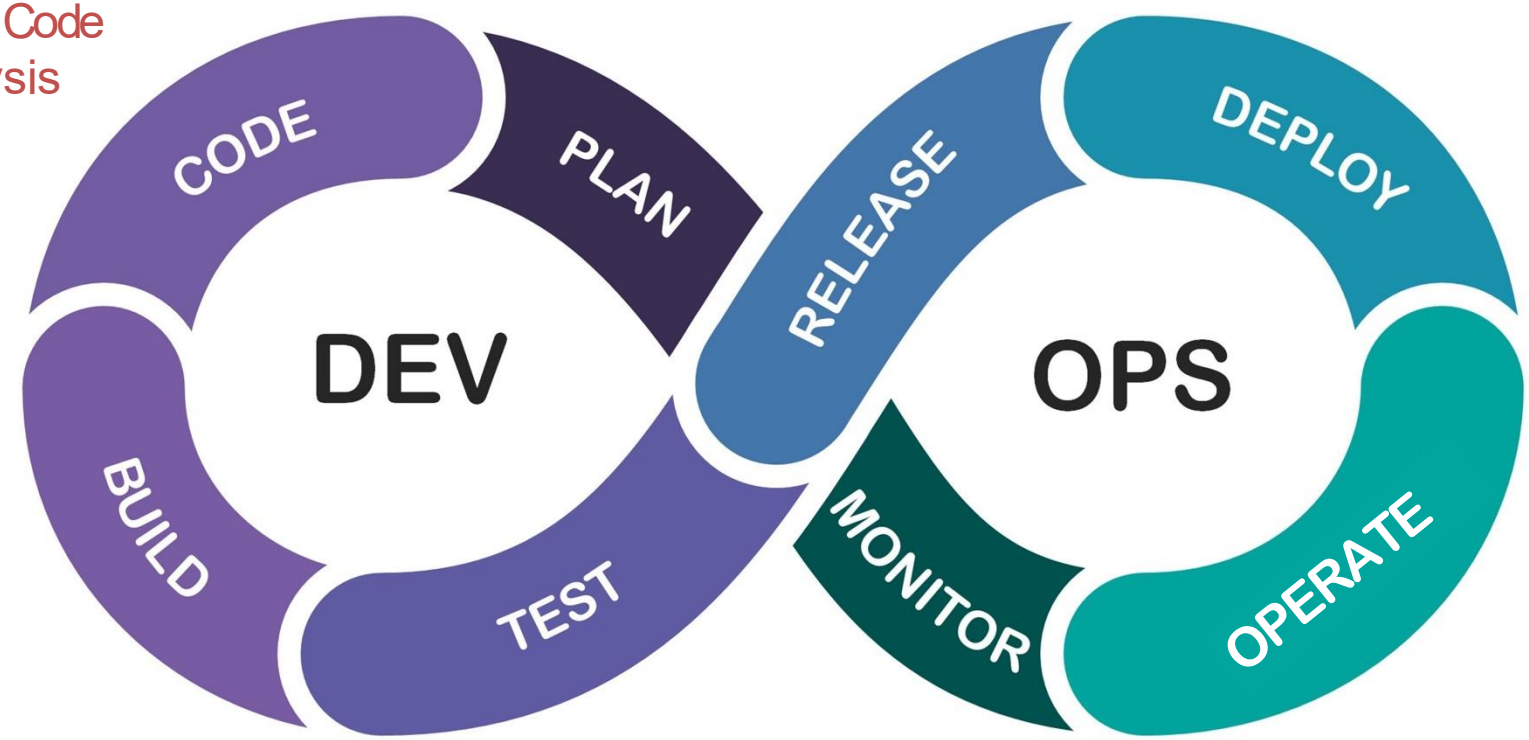
Threat model
Code Standards



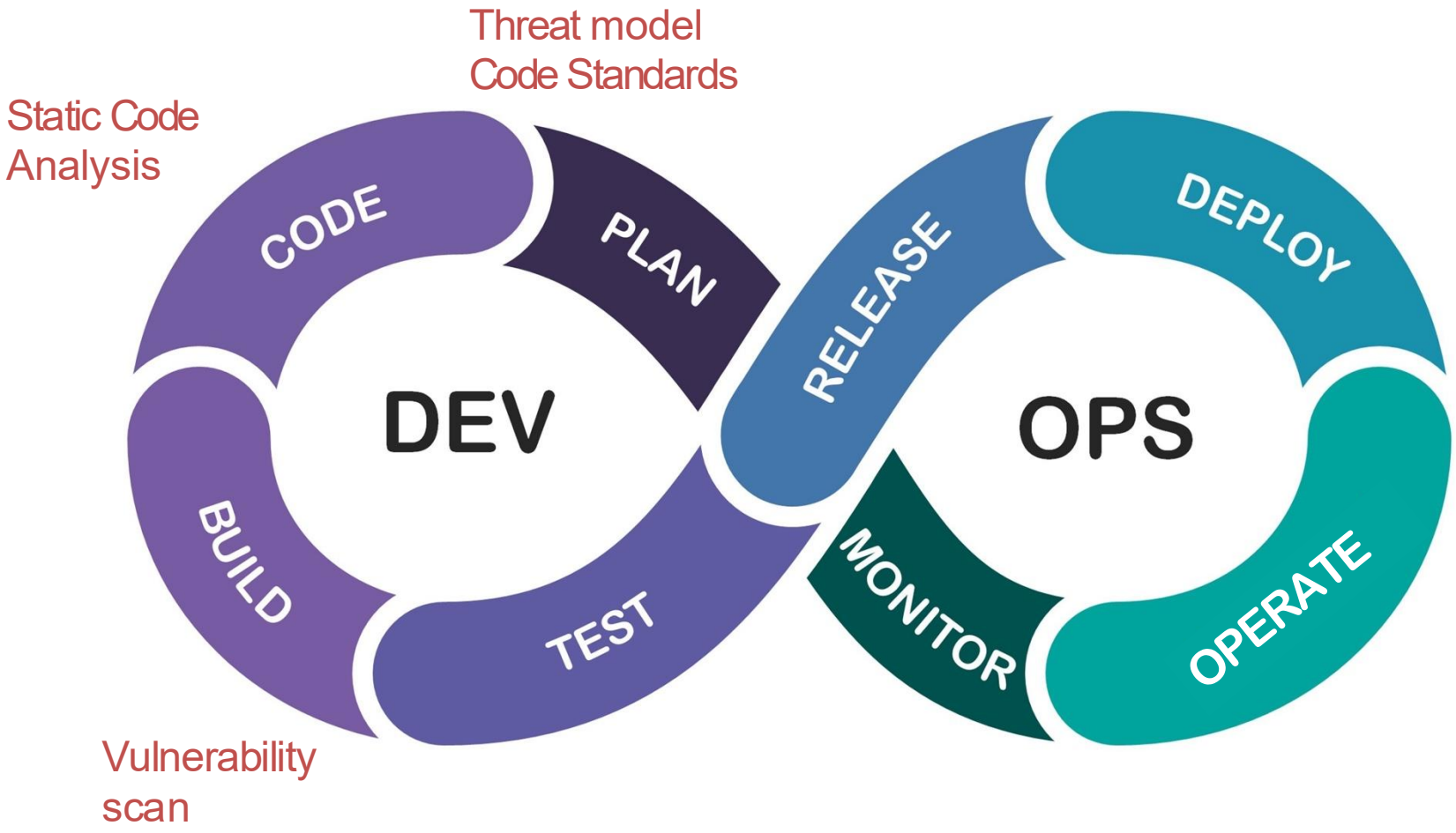
Merge DevOps and Sec

Threat model
Code Standards

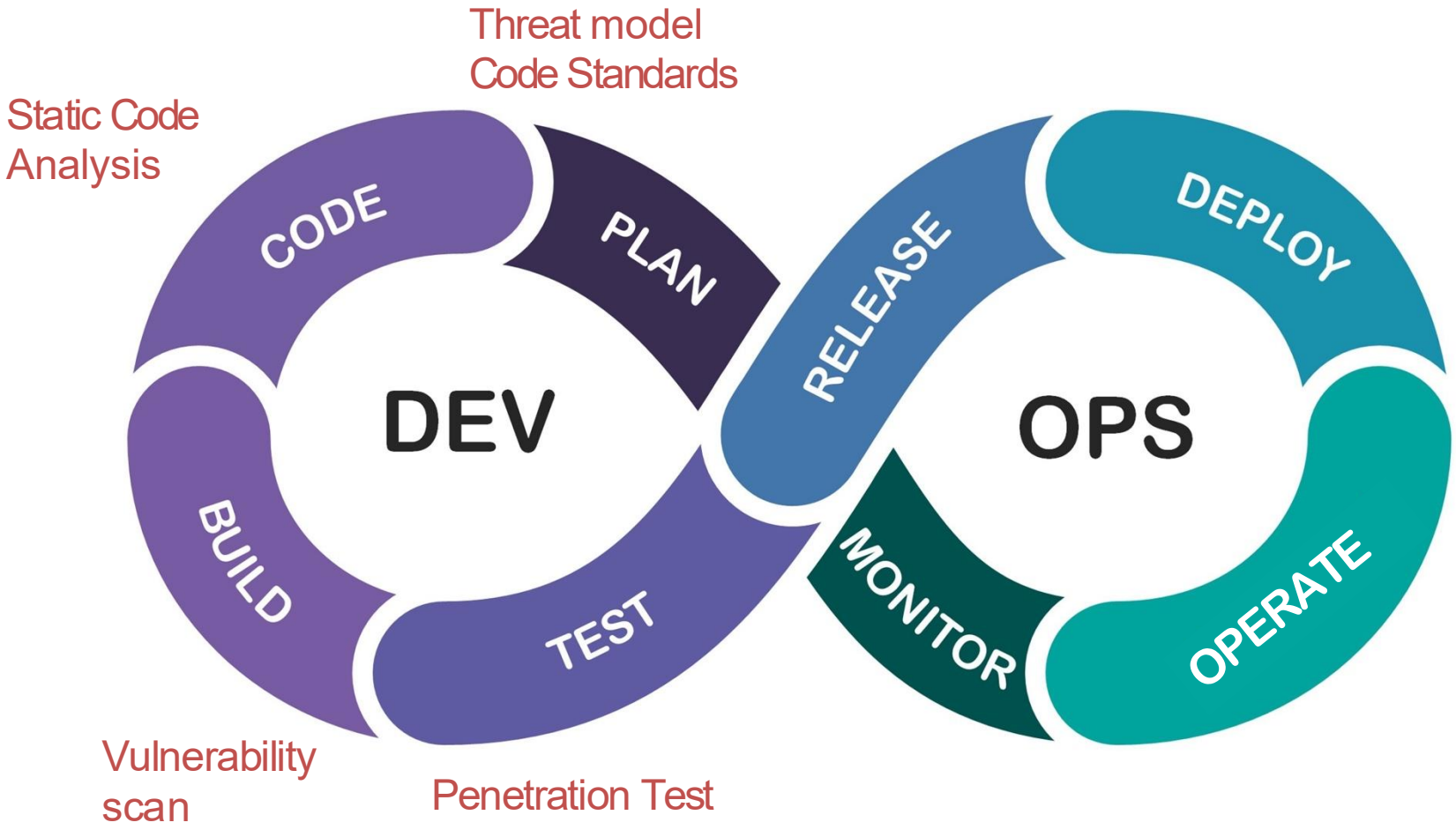
Static Code
Analysis



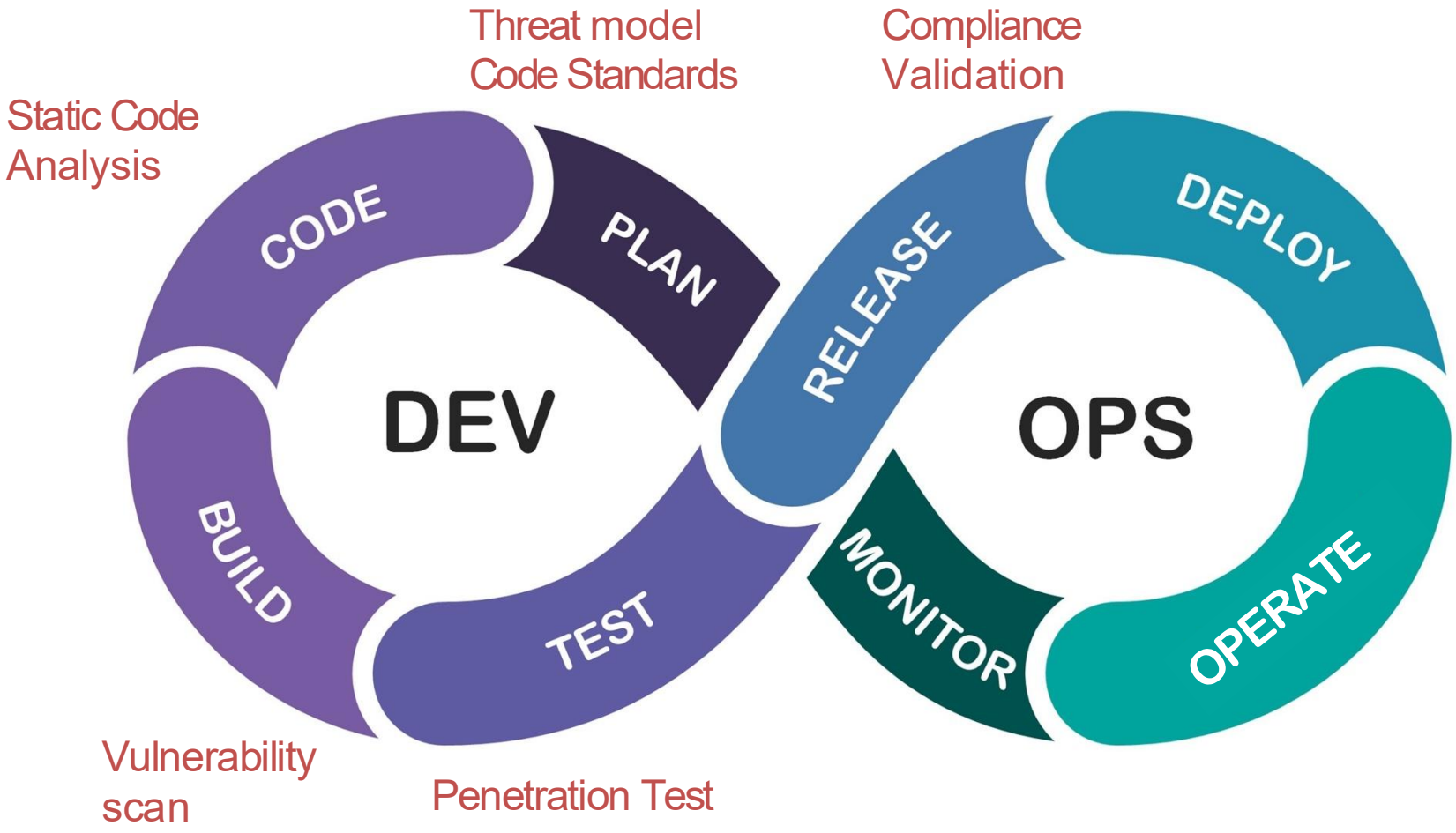
Merge DevOps and Sec



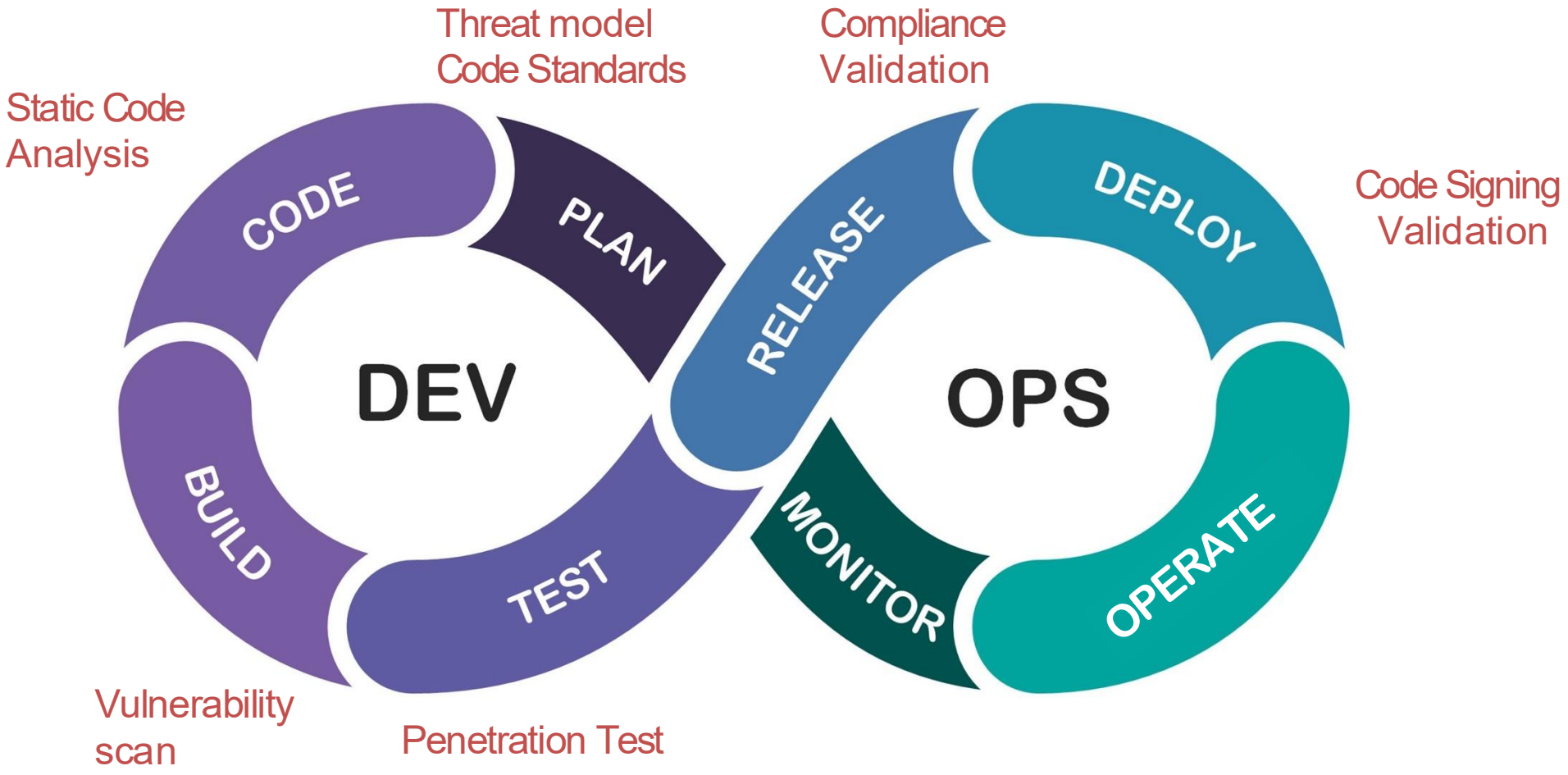
Merge DevOps and Sec



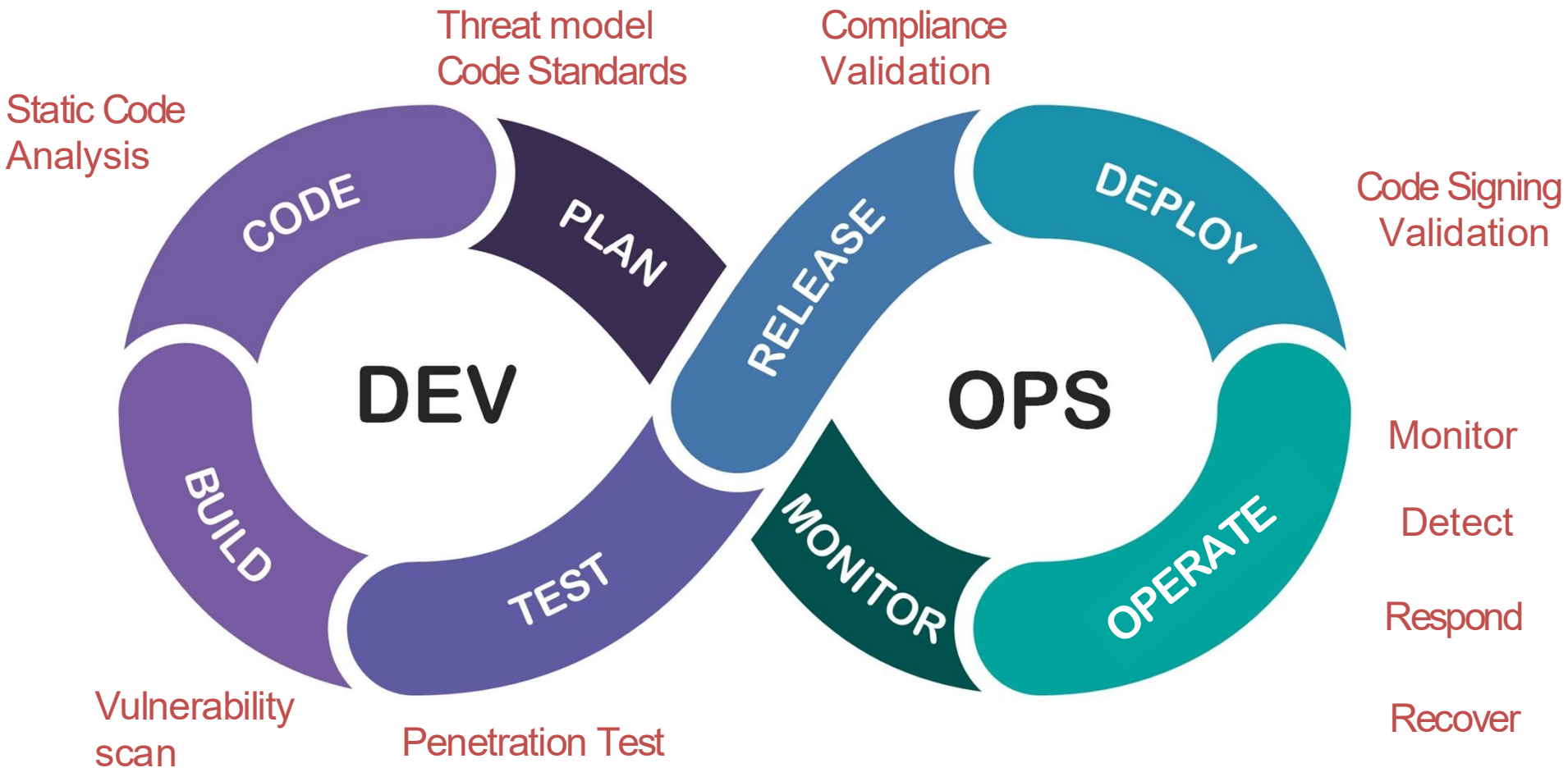
Merge DevOps and Sec



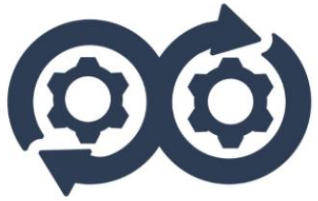
Merge DevOps and Sec



Merge DevOps and Sec



DevSecOps Myths



DevSecOps Myths

We cannot introduce DevSecOps Because…

DevSecOps myths

We cannot introduce DevSecOps Because…

- We need a special DevSecOps team

DevSecOps myths

We cannot introduce DevSecOps Because…

- We need a special DevSecOps team

"DevSecOps is: Creating empowered engineering teams taking ownership of how their product performs all the way to production, including security."

DevSecOps myths

We cannot introduce DevSecOps Because…

- The security team still needs to do all security checks for us.

DevSecOps myths

We cannot introduce DevSecOps Because…

- The security team still needs to do all security checks for us.

By handing over security checks to other teams we introduce delays and time lab into our agile process, instead we should ask the security team to codify their checks so we can build them into our development process automatically.

DevSecOps myths

We cannot introduce DevSecOps Because…

- We don't have enough resources to do DevSecOps, just buy a tool that does it for us.

DevSecOps myths

We cannot introduce DevSecOps Because…

- We don't have enough resources to do DevSecOps, just buy a tool that does it for us.

DevSecOps is not about a capability, it is about a culture, buying a tool is not culture changing. Tools are required to make DevSecOps possible these tools supplement the existing development process and help you deliver DevSecOps, if you have the correct culture in place.

DevSecOps myths

We cannot introduce DevSecOps Because…

- DevSecOps will just slow down our developers.

DevSecOps myths

We cannot introduce DevSecOps Because…

- DevSecOps will just slow down our developers.

DevSecOps is about empowering developers to ensure their product gets to production with appropriate security built-in. Traditional approaches to security required testing after developers complete coding and before deployment to production.

Because this is so late in the lifecycle it takes longer to fix and retest software compared to identifying the issue at an earlier state in the lifecycle, so DevSecOps can save time and increase developer speed.

DevSecOps myths

We cannot introduce DevSecOps Because…

- DevSecOps will result in developers giving up control and won't be able to plan.

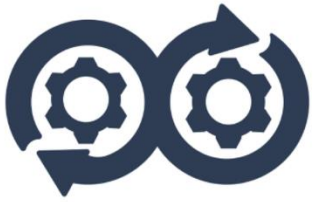
DevSecOps myths

We cannot introduce DevSecOps Because…

- DevSecOps will result in developers giving up control and won't be able to plan.

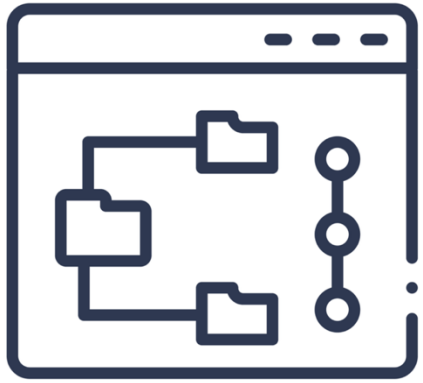
With DevSecOps developers gain control by running security checks at the best possible opportunity to help developers fix the issue quickly and easily. No longer are developers dependent on external teams, and gain control of the work and schedule.

Terraform Introduction



Automating Infrastructure

Provisioning resources

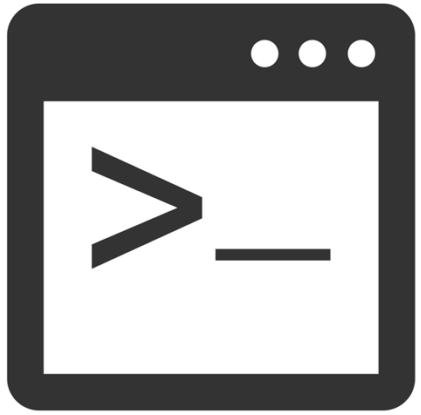


Version Control

Plan Updates

Reusable Templates

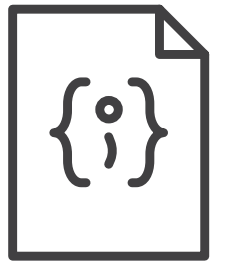
Terraform components



Terraform executable

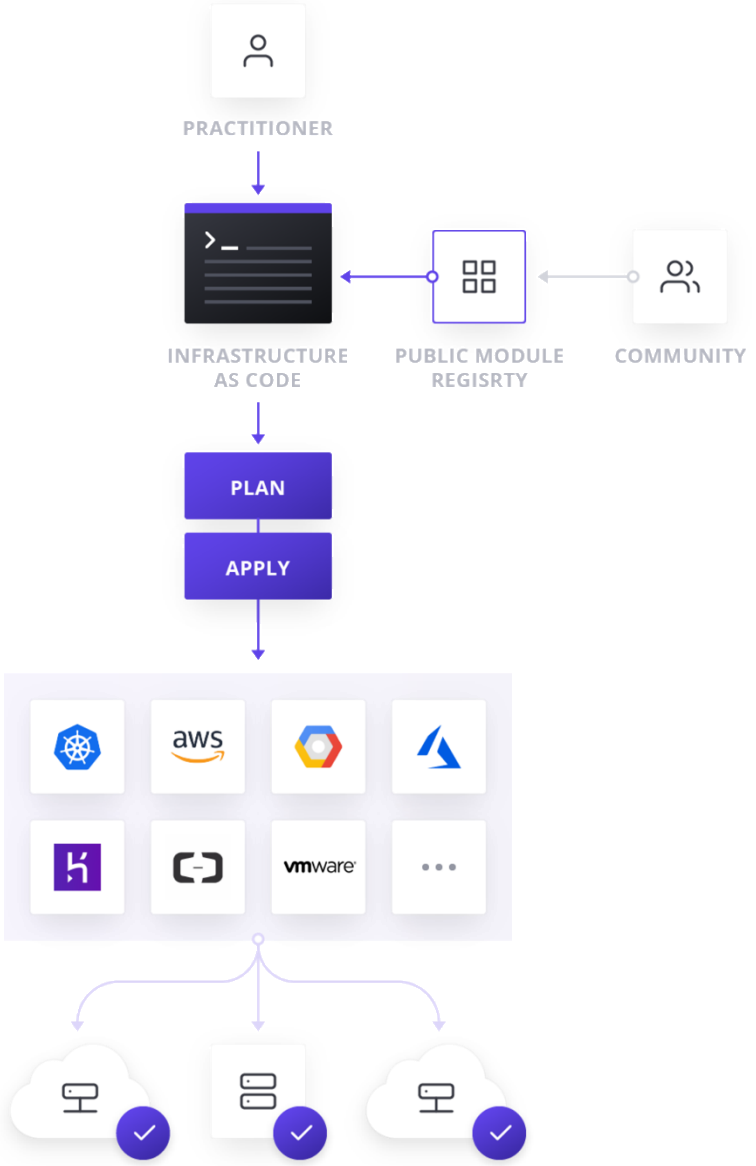
Terraform files

Terraform plugins

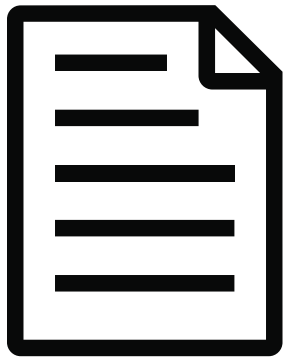


Terraform state

Terraform architecture



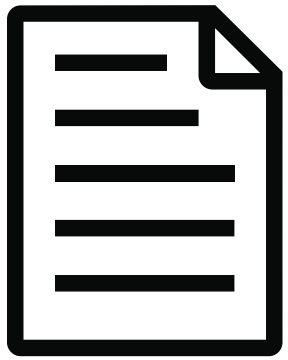
Terraform files



```
##Amazon Infrastructure
provider "aws" {
  region = "${var.aws_region}"
}
```

- Provider defines what infrastructure Terraform will be managing
  - Pass variables to provider
    - Region, Flavor, Credentials

Terraform files



```
data "aws_ami" "ubuntu" {
  most_recent = true
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  owners = ["099720109477"] # Canonical
}
```

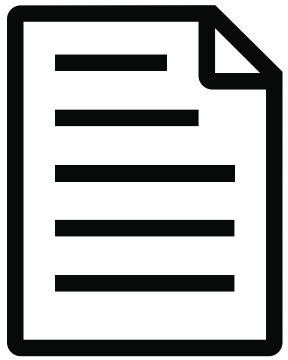
- Data sources
  - Queries the AWS API for the latest Ubuntu 16.04 image.
  - Stores results in a variable which is used later.

Terraform files

```
resource "aws_instance" "aws-k8s-master" {
  subnet_id              = "${var.aws_subnet_id}"
  depends_on             = ["aws_security_group.k8s_sg"]
  ami                    = "${data.aws_ami.ubuntu.id}"
  instance_type          = "${var.aws_instance_size}"
  vpc_security_group_ids = ["${aws_security_group.k8s_sg.id}"]
  key_name               = "${var.aws_key_name}"
  count                  = "${var.aws_master_count}"
  root_block_device {
    volume_type           = "gp2"
    volume_size           = 20
    delete_on_termination = true
  }
  tags {
    Name = "k8s-master-${count.index}"
    role = "k8s-master"
  }
}
```

- Resource: Defines specifications for creation of infrastructure.

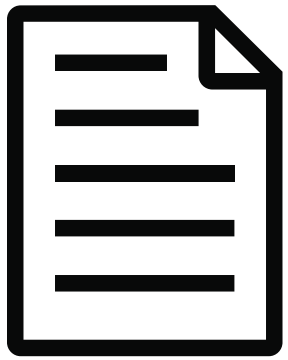
Terraform files



```
resource "aws_instance" "aws-k8s-master" {
  subnet_id              = "${var.aws_subnet_id}"
  depends_on             = ["aws_security_group.k8s_sg"]
  ami                    = "${data.aws_ami.ubuntu.id}"
  instance_type          = "${var.aws_instance_size}"
  vpc_security_group_ids = ["${aws_security_group.k8s_sg.id}"]
  key_name               = "${var.aws_key_name}"
  count                  = "${var.aws_master_count}"
  root_block_device {
    volume_type           = "gp2"
    volume_size           = 20
    delete_on_termination = true
  }
  tags {
    Name = "k8s-master-${count.index}"
    role = "k8s-master"
  }
}
```

- ami is set by results of previous data source query

Terraform files



```
##AWS Specific Vars
variable "aws_master_count" {
  default = 10
}
variable "aws_worker_count" {
  default = 20
}
variable "aws_key_name" {
  default = "k8s"
}
variable "aws_instance_size" {
  default = "t2.small"
}
variable "aws_region" {
  default = "us-west-1"
}
```

- Declare all variables in variables.tf
- OPTIONAL: Define sane defaults in variables.tf

Terraform files

```
resource "aws_instance" "aws-k8s-master" {
  subnet_id              = "${var.aws_subnet_id}"
  depends_on             = ["aws_security_group.k8s_sg"]
  ami                    = "${data.aws_ami.ubuntu.id}"
  instance_type          = "${var.aws_instance_size}"
  vpc_security_group_ids = ["${aws_security_group.k8s_sg.id}"]
  key_name               = "${var.aws_key_name}"
  count                  = "${var.aws_master_count}"
  root_block_device {
    volume_type           = "gp2"
    volume_size           = 20
    delete_on_termination = true
  }
  tags {
    Name = "k8s-master-${count.index}"
    role = "k8s-master"
  }
}
```

- Value is 'count' is setup by variable in variables.tf

POP QUIZ:

What are the primary use cases for Terraform?

POP QUIZ:

What are the primary use cases for Terraform?

- Automating the provisioning and management of cloud and on-premises infrastructure

POP QUIZ:

What are the primary use cases for Terraform?

- Automating the provisioning and management of cloud and on-premises infrastructure
- Ensuring consistent configurations across development, testing, and production environments

POP QUIZ:

What are the primary use cases for Terraform?

- Automating the provisioning and management of cloud and on-premises infrastructure
- Ensuring consistent configurations across development, testing, and production environments
- Enabling repeatable and version-controlled infrastructure deployments through code

Terraform CLI

Run `terraform` command

Output:

```
Usage: terraform [-version] [-help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform
interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
    get                Download and install modules for the
configuration
    graph              Create a visual graph of Terraform
resources
```

Terraform CLI

Command:

```
terraform init
```

Output:

```
Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "docker"...
```

Terraform fetches any required providers and modules and stores them in the .terraform directory. Check it out and you'll see a plugins folder.

Terraform CLI

Command:

```
terraform validate
```

Validate all of the Terraform files in the current directory. Validation includes basic syntax check as well as all variables declared in the configuration are specified.

Terraform CLI

Command:

```
terraform plan
```

Output:

```
Terraform will perform the following actions:

  + aws_instance.aws-k8s-master
      id:                                   <computed>
      ami:                                  "ami-01b45..."
      instance_type:                        "t3.small"
```

Plan is used to show what Terraform will do if applied. It is a dry run and does not make any changes.

Terraform CLI

Command:

```
terraform apply
```

Output:

```
❯ terraform apply "rapid-app.out"
aws_security_group.k8s_sg: Creating...
  arn:                                      "" => "<computed>"
  description:                              "" => "Allow all
inbound traffic necessary for k8s"
  egress.#:                                 "" => "1"
  egress.482069346.cidr_blocks.#:           "" => "1"
  egress.482069346.cidr_blocks.0:           "" => "0.0.0.0/0"
```

Performs the actions defined in the plan

Terraform CLI

Command:

```
terraform state show <resource>
```

```
terraform state show aws_instance.lab2-tf-example
```

Output:

```
ami                                  = "ami-830c94e3"
availability_zone                    = "us-west-2b"
cpu_core_count                       = 1
...
```

Shows information about provided resource.

Terraform CLI

The `terraform refresh` command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. This can be used to detect any drift from the last-known state, and to update the state file.
This does not modify infrastructure but does modify the state file. If the state is changed, this may cause changes to occur during the next `plan` or `apply`.

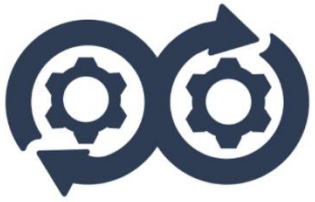
Terraform CLI

Command:

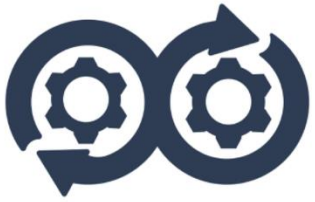
```
terraform destroy [-auto-approve]
```

Destroys all the resources in the state file.
  -auto-approve (don't prompt for confirmation)

Lab: Setup VM



Lab: Build first instance



Terraform CLI

Terraform is normally run from inside the directory containing the *.tf files for the root module. Terraform checks that directory and automatically executes them.

In some cases, it makes sense to run the Terraform commands from a different directory. This is true when wrapping Terraform with automation. To support that Terraform can use the global option – `chdir=...` which can be included before the name of the subcommand.

Terraform CLI

The `chdir` option instructs Terraform to change its working directory to the given directory before running the given subcommand. This means that any files that Terraform would normally read or write in the current working directory will be read or written in the given directory instead.

Terraform CLI

Command:

```
terraform -chdir=environments/dev (apply|plan|destroy)
```

Output:

```
❯ terraform apply
aws_security_group.k8s_sg: Creating...
  arn:                                      "" => "<computed>"
  description:                              "" => "Allow all
inbound traffic necessary for k8s"
  egress.#:                                 "" => "1"
  egress.482069346.cidr_blocks.#:           "" => "1"
  egress.482069346.cidr_blocks.0:           "" => "0.0.0.0/0"
```

Performs the subcommand in the specified directory.

Terraform CLI

It can be time consuming to update a configuration file and run `terraform apply` repeatedly to troubleshoot expressions not working.

Terraform has a console subcommand that provides an interactive console for evaluating these expressions.

Terraform CLI

```
variable "x" {

default = [
        {
          name = "first",
          condition = {
            age = "1"
          }
          action = {
            type = "Delete"
          }
        },{
          name = "second",
          condition = {
            age = "2"
          }
          action = {
            type = "Delete"
          }
      }
    ]
}
```

Terraform CLI

Command:

```
terraform console
```

Output:

```
> var.x[1].name
"second"

var.x[0].condition
{
  "age" = "1"
}
```

Test expressions and interpolations interactively.

Terraform CLI

Terraform includes a graph command for generating visual representation of the configuration or execution plan. The output is in DOT format, which can be used by GraphViz to generate charts.

Terraform CLI

Command:

```
terraform graph
```

Output:

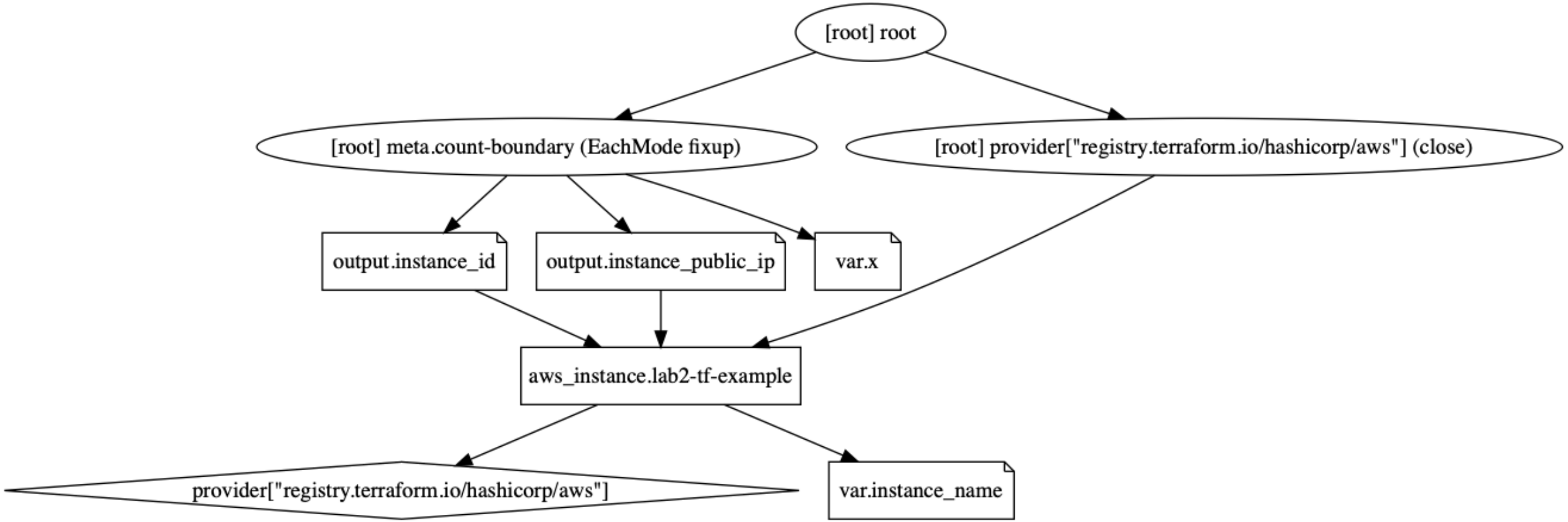
```
digraph {
        compound = "true"
        newrank = "true"
        subgraph "root" {
                "[root] aws_instance.lab2-tf-example (expand)"
[label = "aws_instance.lab2-tf-example", shape = "box"]
...
```

Create a visual graph of Terraform resources

```
terraform graph | dot -Tsvg > graph.svg
```

Terraform CLI

```
terraform graph| dot -Tsvg > graph.svg
```



Terraform vs JSON

ARM JSON:

```
"name" : "[concat(parameters('PilotServerName'), '3')]",
```

Terraform:

```
name = "${var.PilotServerName}3"
```

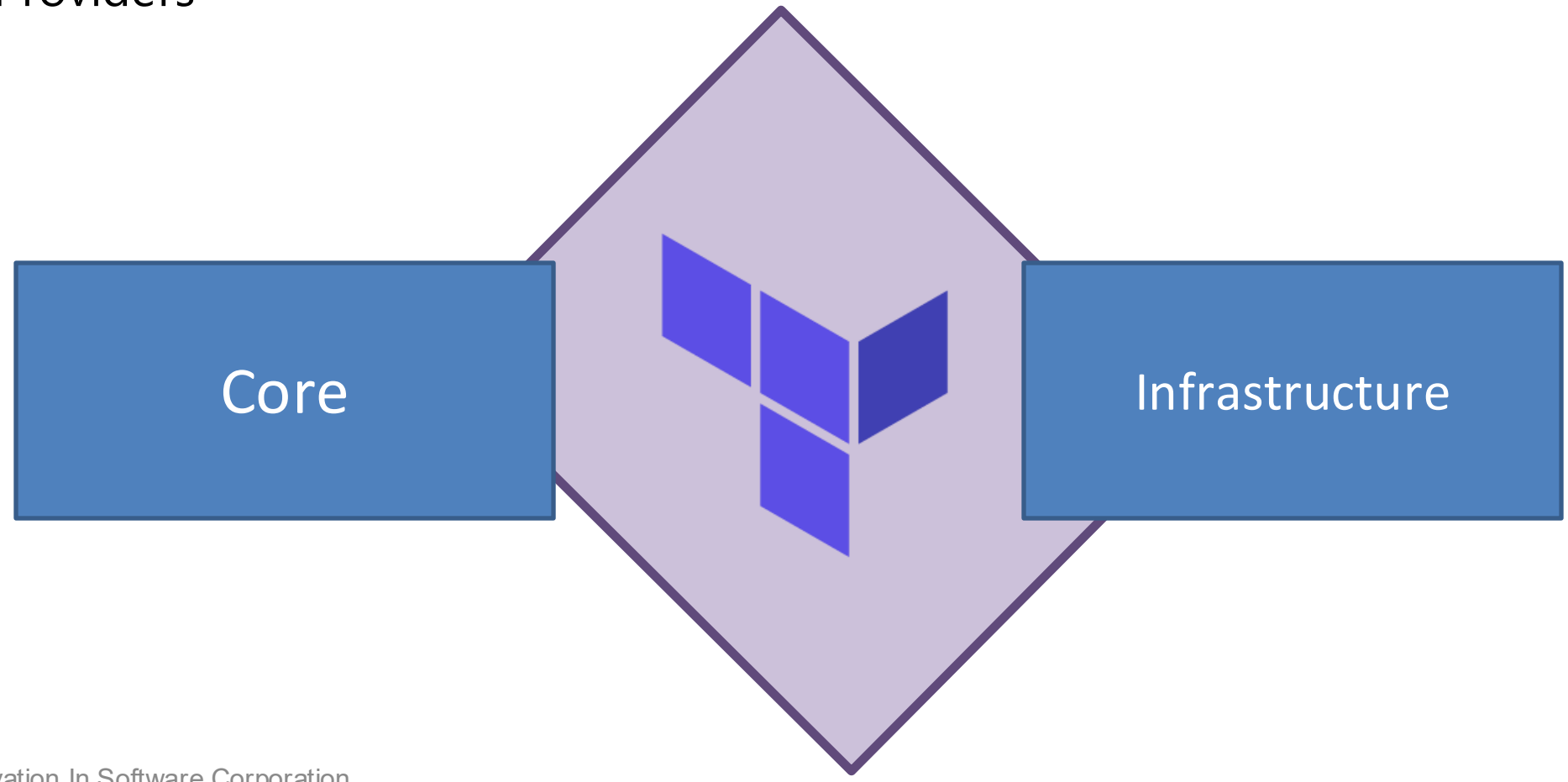
Terraform code (HCL) is "easy" to learn and "easy" to read. It is also more compact than equivalent JSON configuration.

Terraform model

Terraform code is broken into three parts: Core Terraform, Modules and Providers

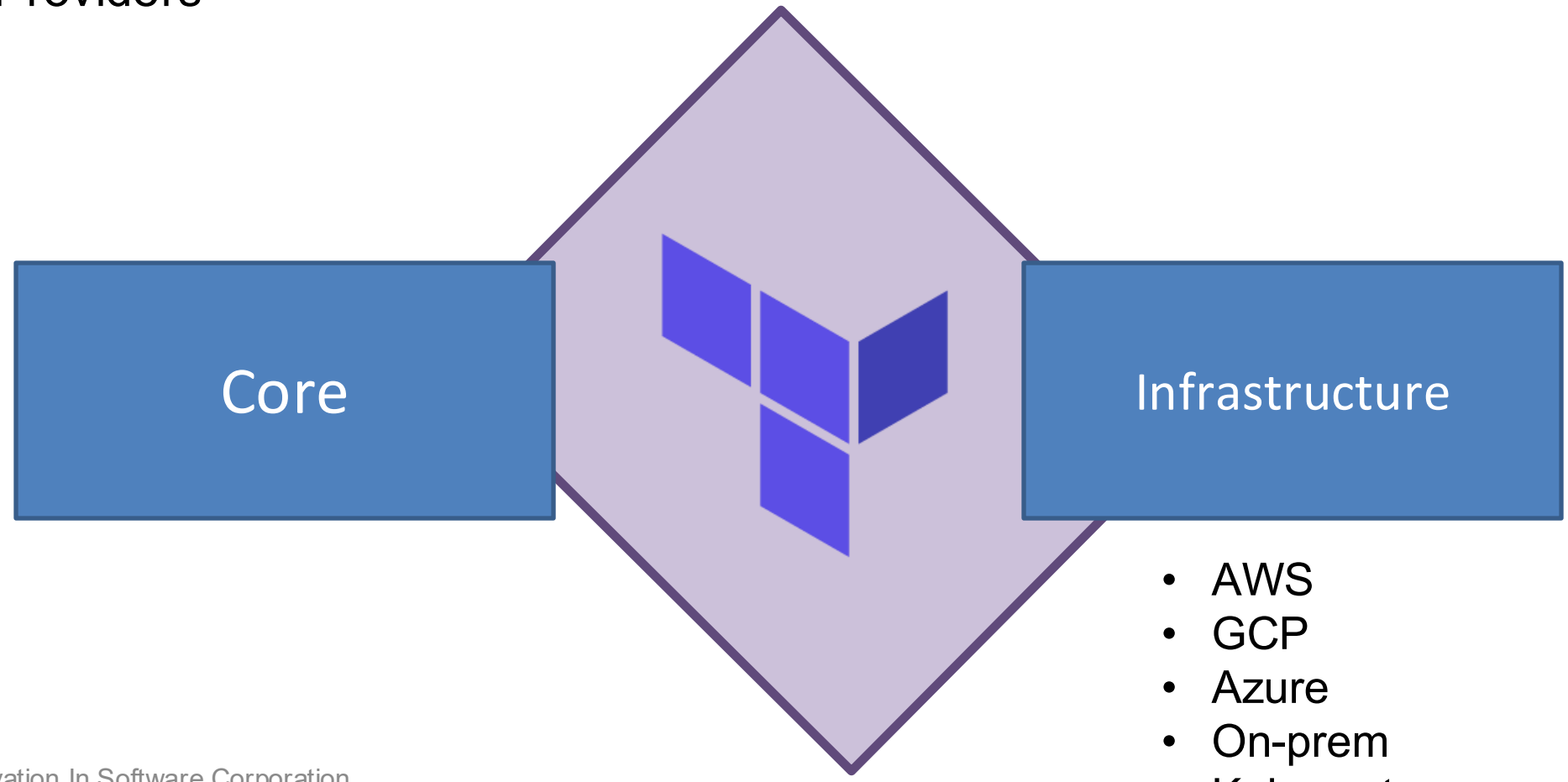
Terraform model

Terraform code is broken into three parts: Core Terraform, Modules and Providers



Terraform model

Terraform code is broken into three parts: Core Terraform, Modules and Providers



- AWS
- GCP
- Azure
- On-prem
- Kubernetes

Configuration files

Terraform gives you flexibility in how you organize your code. You can structure your files modularly—or keep everything in one place. Terraform doesn't force you into a specific design.

`provider.tf` – Defines the provider configuration

`main.tf` – Contains the resource definitions.

`variables.tf` – Declares input variables used in `main.tf`

Terraform files

- core = Terraform language, logic, and tooling.
- provider = Pluggable code to allow Terraform to talk to vendor APIs
- modules = A container for multiple resources that are used together.

Terraform providers

Terraform relies on plugins called "providers" to interface with remote systems.

Terraform configurations must declare which providers they require so that Terraform can install and use them. Additionally, some providers require configuration (like endpoint URLs or cloud regions) before they can be used.

Terraform providers

Each provider adds resources and/or data sources that Terraform manages.

Every resource type is implemented by a provider. Terraform can't manage any kind of infrastructure without providers.

Providers are used to configure infrastructure platforms (either cloud or self-hosted). Providers also offer utilities for tasks like generating random numbers for unique resource names.

Terraform files

```
##Amazon Infrastructure
provider "aws" {
  region = "${var.aws_region}"
}
```

- Provider defines what infrastructure Terraform will be managing
  - Pass variables to provider
    - Region, Flavor, Credentials

Terraform provider configuration

```
# The default provider configuration; resources that begin with `aws_` will use
# it as the default, and it can be referenced as `aws`.
provider "aws" {
  region = "us-east-1"
}

# Additional provider configuration for west coast region; resources can
# reference this as `aws.west`.
provider "aws" {
  alias  = "west"
  region = "us-west-2"
}
```

Terraform supports 'aliases' for multiple configurations for the same provider, and you can select which one to use on a per-resource or per-module basis.

Terraform provider configuration

```
provider "azurerm" {
  version = "=1.30.1"
}
```

Terraform allows you to configure the providers in code. configure specific versions or pull in latest.

Terraform provider configuration

```
provider "azurerm" {
  version = "=1.30.1"
  subscription_id = "SUBSCRIPTION-ID"
  client_id       = "CLIENT-ID"
  client_secret   = "CLIENT-SECRET"
  tenant_id       = "TENANT_ID"
}
```

Providers allow you to authenticate in the code block.
This is a very BAD idea.

Terraform provider configuration

For Azure that can be `az login`, Managed Service Identity, or environment variables.

```
az login
```

```
export ARM_TENANT_ID=
export ARM_SUBSCRIPTION_ID=
export ARM_CLIENT_ID=
export ARM_CLIENT_SECRET=
```

Terraform provider configuration

Terraform will use the native tool's method of authentication. Environment variables, shared credentials files, or static credentials.

```
provider "aws" {}
```

```
export AWS_ACCESS_KEY_ID=
export AWS_SECRET_ACCESS_KEY=
```

Terraform provider configuration

Terraform will use the native tool's method of authentication. Environment variables, shared credentials files, or static credentials.

```
provider "aws" {
  region                   = "us-west-2"
  shared_credentials_file  = "~/.aws/creds"
  profile                  = "customprofile"
}
```

Terraform provider configuration

Terraform will use the native tool's method of authentication. Environment variables, shared credentials files, or static credentials.

```
provider "aws" {
  access_key     = "MYKEY"
  secret_key     = "MYSECRET"
}
```

Terraform ASA provider

```
provider "ciscoasa" {
  api_url           = "https://10.0.0.5"
  username          = "admin"
  password          = "YOUR SECRET PASSWORD"
  ssl_no_verify     = false
}
```

Providers allow you to authenticate in the code block.
This is a very BAD idea.
Use environment variables:
`CISCOASA_USER`
`CISCOASA_PASSWORD`

Terraform resources

Resources are the most important element of the Terraform language. Each resource block describes one or more infrastructure objects, such as networks, instances, or higher-level objects such as DNS records.

Terraform resources

A resource block declares a resource of a given type ("aws_instance") with a given local name ("web"). The name is used to refer to this resource from elsewhere in the same Terraform module but has no significance outside that module's scope.

The resource type and name together serve as an identifier for a given resource and so must be unique within a module.

Terraform resources

Every Terraform resource is structured the exact same way.

```
resource "type" "name" {
  parameter = "foo"
  parameter2 = "bar"
  list = ["one", "two", "three"]
```

resource = top level keyword

Terraform resources

Within the block body (between { and }) are the configuration arguments for the resource itself. Most arguments in this section depend on the resource type, and indeed in this example both ami and instance_type are arguments defined specifically for the aws_instance resource type.

Terraform resources

Every Terraform resource is structured the exact same way.

```
resource "type" "name" {
  parameter = "foo"
  parameter2 = "bar"
  list = ["one", "two", "three"]
```

type = this is the name of the resource. The first part tells you which provider is belongs to. Example: `azurerm_virtual_machine`. This means the provider is Azure and the specific type of resources is a virtual machine.

Terraform resources

Every Terraform resource is structured the exact same way.

```
resource "type" "name" {
  parameter = "foo"
  parameter2 = "bar"
  list = ["one", "two", "three"]
```

name = arbitrary name to refer to resource. Used internally by TF and cannot be a variable.

Terraform ASA ACL resource

```
resource "ciscoasa_acl" "foo" {
  name = "aclname"
  rule {
    source                = "192.168.10.5/32"
    destination           = "192.168.15.0/25"
    destination_service = "tcp/443"
  }
  rule {
    source                = "192.168.10.0/24"
```

Example of creating ACL

Terraform ASA static route resource

```
resource "ciscoasa_static_route" "ipv4_static_route" {
  interface               = "inside"
  network                 = "10.254.0.0/16"
  gateway                 = "192.168.10.20"
  }

resource "ciscoasa_static_route" "ipv6_static_route" {
  interface               = "inside"
  network                 = "fd01:1337::/64"
  gateway                 = "fd01:1338::1"
```

Resources (building blocks)

Create an Azure Resource Group.

- Azure requires all resources to be assigned to a resource group.

```
resource "azurerm_resource_group" "training" {
  name     = "training-workshop"
  location = "westus"
}
```

Resources (building blocks)

TIP: You can assign random names to resources.

```
resource "random_id" "project_name" {
  byte_length = 4
}
resource "azurerm_resource_group" "training" {
  name       = "${random_id.project_name.hex}-training"
  location   = "westus"
}
```

Terraform resources

Some resource types provide a special timeouts nested block argument that allows you to customize how long certain operations are allowed to take before being considered to have failed. For example, aws_db_instance allows configurable timeouts for create, update and delete operations.

Terraform resources

Timeouts are handled entirely by the resource type implementation in the provider, but resource types offering these features follow the convention of defining a child block called timeouts that has a nested argument named after each operation that has a configurable timeout value. Each of these arguments takes a string representation of a duration, such as "60m" for 60 minutes, "10s" for ten seconds, or "2h" for two hours.

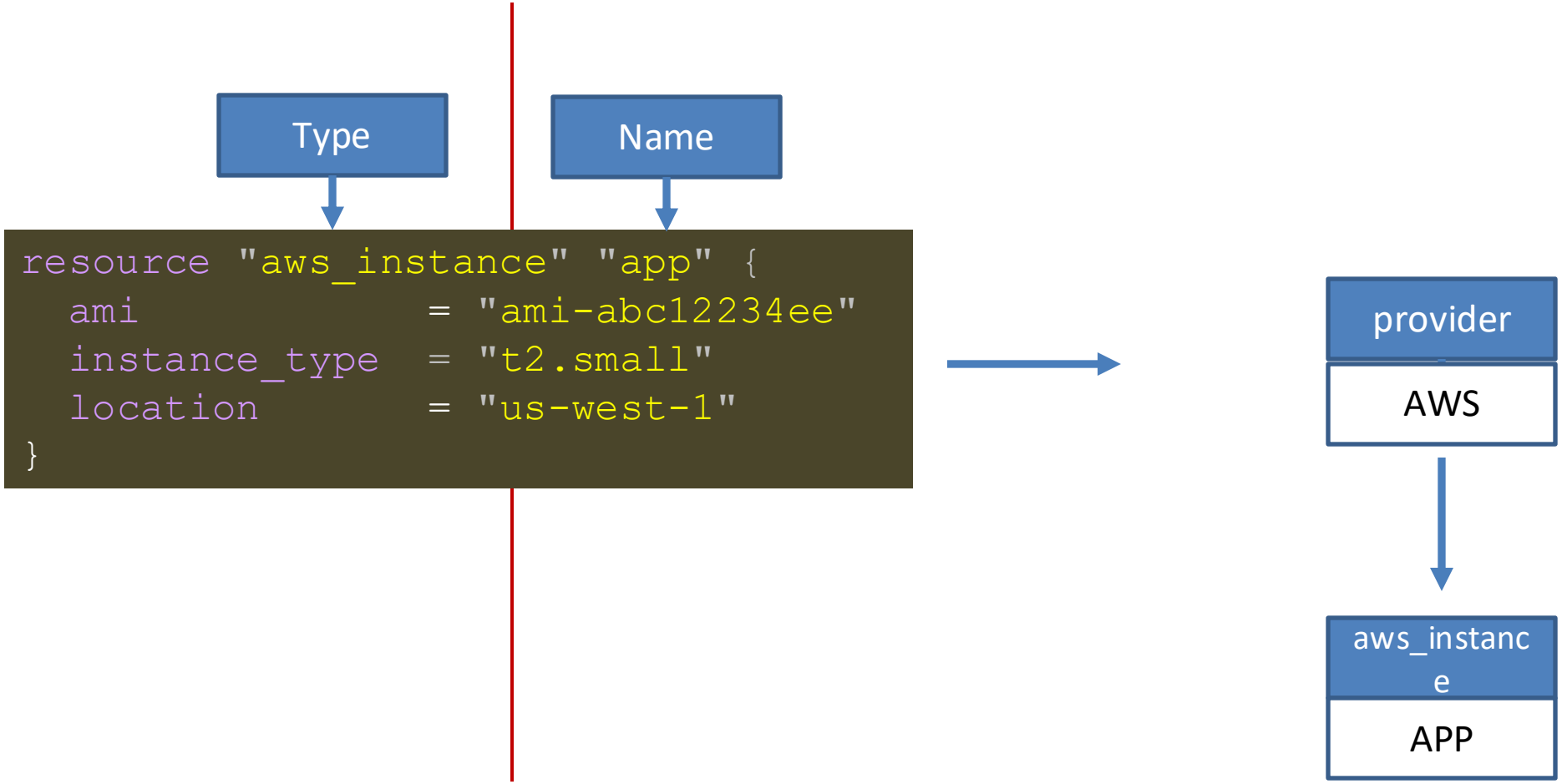
Terraform resources

```
resource "aws_db_instance" "example" {
  # ...

  timeouts {
    create = "60m"
    delete = "2h"
  }
}
```

Timeout arguments

Resources (building blocks)



Terraform local-only resources

While most resource types correspond to an infrastructure object type that is managed via a remote network API, there are certain specialized resource types that operate only within Terraform itself, calculating some results and saving those results in the state for future use.

For example, you can use local-only resources for:

- Generating private keys
- Issue self-signed TLS certs
- Generating random ids

The behavior of local-only resources is the same as all other resources, but their result data exists only within the Terraform state. "Destroying" such a resource means only to remove it from the state, discarding its data.

Terraform variables

Input variables serve as parameters for a Terraform module, allowing aspects of the module to be customized without altering the module's own source code, and allowing modules to be shared between different configurations.

When you declare variables in the root module of your configuration, you can set their values using CLI options and environment variables. When you declare them in child modules, the calling module should pass values in the module block.

Terraform variables

Each input variable accepted by a module must be declared using a variable block.

The label after the variable keyword is a name for the variable, which must be unique among all variables in the same module. This name is used to assign a value to the variable from outside and to reference the variable's value from within the module.

```
variable "image_id" {
  type = string
}

variable "availability_zone_names" {
  type    = list(string)
  default = ["us-west-1a"]
}

variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  default = [
    {
      internal = 8300
      external = 8300
      protocol = "tcp"
    }
  ]
}
```

Terraform variables

Terraform has some reserved variables:

- source
- version
- providers
- count
- for_each
- lifecycle
- depends_on
- locals

- These are reserved for meta-arguments in module blocks.

```
variable "image_id" {
  type = string
}

variable "availability_zone_names" {
  type    = list(string)
  default = ["us-west-1a"]
}

variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  default = [
    {
      internal = 8300
      external = 8300
      protocol = "tcp"
    }
  ]
}
```

Terraform variables

Optional variable arguments:

- default: A default value which makes the variable optional
- type: Specifies value type accepted for the variable.
- description: Specifies the variable's documentation
- validation: A block to define validation rules
- sensitive: Does not show value in output

Terraform variables

Set default values for variables in `variables.tf`
If default values are omitted and not set anywhere else, the user will be prompted to provide them.

```
##AWS Specific Vars
variable "aws_master_count" {
  default = 10
}
variable "aws_worker_count" {
  default = 20
}
variable "aws_key_name" {
  default = "k8s"
}
```

Terraform variables (type constraint)

Type constraints are optional but highly encouraged. They can serve as reminders for users of the module and help Terraform return helpful errors if the wrong type is used.

The type argument allows you to restrict acceptable value types. If no type constraint is defined, then a value of any type is accepted.

The keyword any may be used to indicate that any type is acceptable.

If both the type and default arguments are specified, the given default value must be convertible to the specified type.

Terraform variables (type constraint)

Type constraints are created from a mixture of type keywords and constructors.

Supported type keywords:
- string
- number
- bool

Type constructors allow you to specify complex types such as collections:
- list (<TYPE>)
- set (<TYPE>
- map (<TYPE>)
- object ({<ATTR NAME> = <TYPE>, ...})
- tuple ([<TYPE>, ...])

Terraform variables

Using the optional description argument, you can briefly describe the purpose of each variable.

```
## Variable description example
variable "image_id" {
  type = string
  description = "The id of the machine image (AMI)"
}
```

Terraform variables (validation)

In addition to Type Constraints, you can specify arbitrary custom validation rules for a variable.

```
## Variable validation example
variable "image_id" {
  type = string
  description = "The id of the machine image (AMI)"


  validation {
    condition = length(var.image_id) > 4 && substr(var.image_id, 0, 4) == "ami-"
     error_message = "The image_id value must be a valid AMI id, starting with
\"ami-\"."
  }
}
```

Terraform variables (validation)

A condition argument is an expression that must use the value of the variable to return true or false.

The expression can refer only to the variable that the condition applies to and must not produce errors.

Terraform variables

If the failure of an expression is the basis of the validation decision, use the can function to detect such errors. For example:

```
## Variable validation example
variable "image_id" {
  type = string
  description = "The id of the machine image (AMI)"


  validation {
    # regex(...) fails if it cannot find a match
    condition = can(regex("^ami-", var.image_id))
    error_message = "The image_id value must be valid."
```

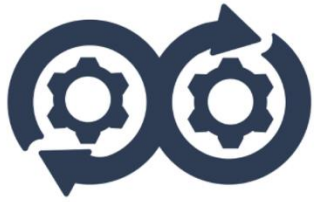
Terraform local values

Local values can hide the actual values being used.
Use local values in moderation. They are helpful to avoid repeating the same values or expressions multiple times in a configuration, but if use too often can make a configuration hard to read by future maintainers.

Only use local values in situations where a single value or result is used in many places AND that value is likely to be changed in the future.

The ability to change the value in one location is the key advantage of local values.

Lab: Variables and output



Lab: Create multi-resource infrastructure