Automate Infrastructure

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 the benefits of DevSecOps
- Use Git for version control
- Describe what Configuration Drift is and how to avoid it using Infrastructure as Code
- Use Terraform to automate infrastructure management.
- Build Ansible playbooks to install and configure applications.
- Build pipelines using GitHub Actions to automate infrastructure and application deployment.

Meet your Instructor: Antoine Victor

- Technical Trainer | DevOps Consultant | Content Creator
- Instructor with 20+ years of Technical Training experience
- Passionate about Teaching by Doing
- Favorite Tools: VS Code, Terraform, Ansible, Azure, AWS, Jenkins, Jira, and more
- Known for hands-on labs, real-world demos, and interactive workshops
- MCSD, MCSE, MCSA CSA, MCT
- Certified Scrum Developer (CSD)
- Certified Scrum Master / Product Owner

- Based in Seattle, WA | Loves Snowboarding, Evs and Renewable Energy
- Drives a Tesla Model S | Enjoys making travel content with his dog Winston

Expertise

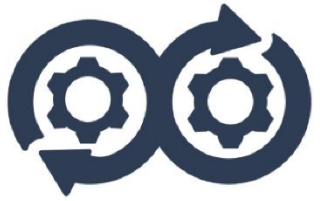
- Agile
- DevOps
- Cloud
- Automation
- CICD
- Full Stack Development

antoine@innovationinsoftware.com
https://github.com/ProDataMan
https://www.linkedin.com/in/antoinevictor/

Introductions

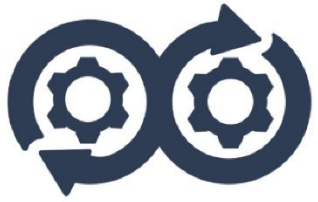
- Name
- Job Role
- Your experience with (scale 1 – 5):
  - Ansible
  - Terraform
  - Git
  - GitHub Actions
- Expectations for course (please be specific)

https://jruels.github.io/gh-auto-infra

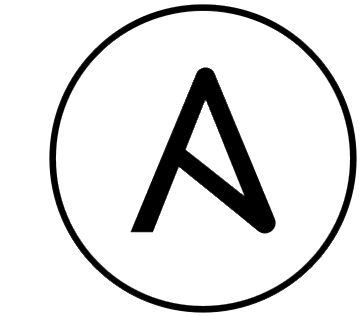


Automation:

Repeatability by automating activities.



Automation tools



ANSIBLE

CHEF

puppet

HashiCorp Terraform

Infrastructure as Code

What is IaC?

Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files
Used with bare-metal as well as virtual machines and many other resources.
Normally a declarative approach

Infrastructure as Code

- Programmatically provision and configure components
- Treat like any other code base
  - Version control
  - Automated testing
  - data backup

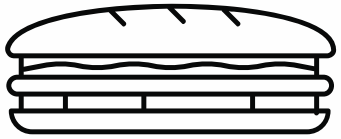
Infrastructure as Code

Provisioning infrastructure through software to achieve consistent and predictable environments.

Core Concepts

- Defined in code
- Stored in source control
- Declarative or imperative

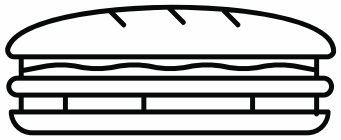
Imperative



Make a sandwich
get bread
get mayo
get turkey
get lettuce

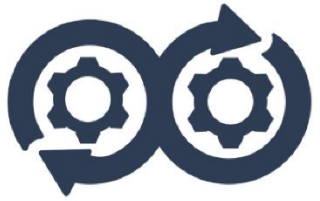
spread mayo on bread
put lettuce in between bread
put turkey in between bread on top of turkey

Declarative



```
# Make a sandwich
food sandwich "turkey" {
  ingredients = [
    "bread", "turkey", "mayo", "lettuce"
  ]
}
```

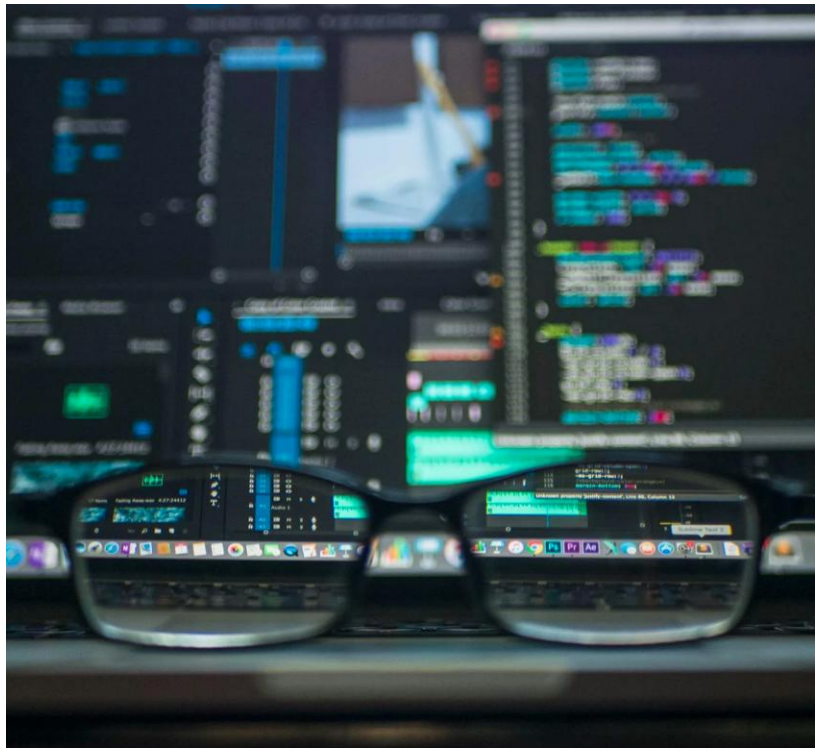
DevSecOps and Policy-as-Code



Introduction to DevSecOps

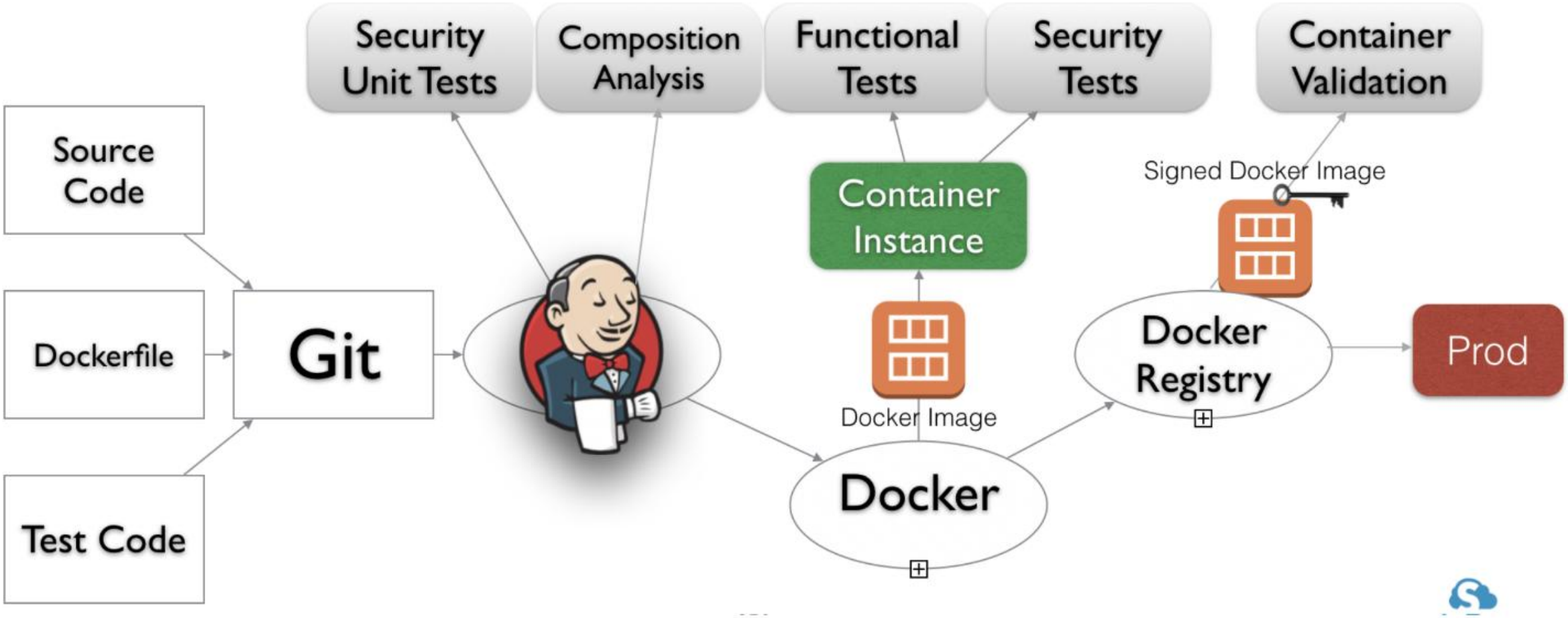
Continuously assessing security measures within the development cycle to gauge their effectiveness and foster ongoing improvement.

What is DevSecOps?

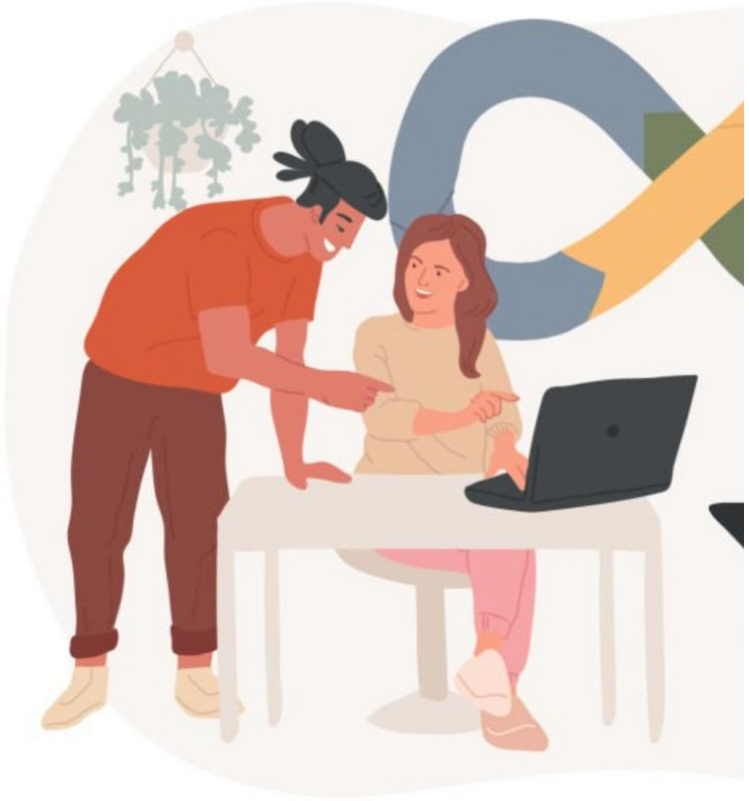


- **DevSecOps** integrates security practices into DevOps, making security a shared responsibility throughout the software development lifecycle.
- **Security** is embedded at every stage, from code development to deployment.
- **Goal:** Catch vulnerabilities early and ensure continuous protection.

Example DevSecOps Architecture



Key principles of DevSecOps



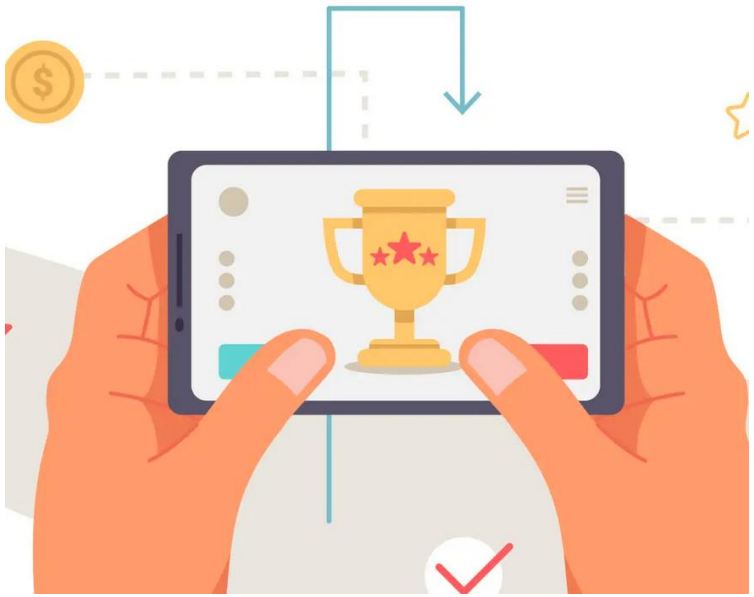
- **Continuous security integration**, automating security checks for faster, safer releases.
- **Close collaboration** among development, security, and operations teams for proactive threat mitigation.
- **Real-time monitoring and feedback loops**, allowing teams to assess and improve security measures based on actual performance.

Objective: To enable fast, secure software delivery by making security an integral part of the development pipeline.

Why DevSecOps Matters

- **Increasing Cyber Threats:** As cyber threats grow more sophisticated, integrating security early is essential to protect data and systems.
- **Faster Development Cycles:** Traditional security practices can slow down the pace; DevSecOps enables rapid, secure releases.
- **Cost Efficiency:** Catching vulnerabilities early reduces the cost of fixing issues post-deployment.
- **Compliance & Trust:** Automated checks ensure that applications meet regulatory standards, increasing reliability and trust.

Benefits of DevSecOps

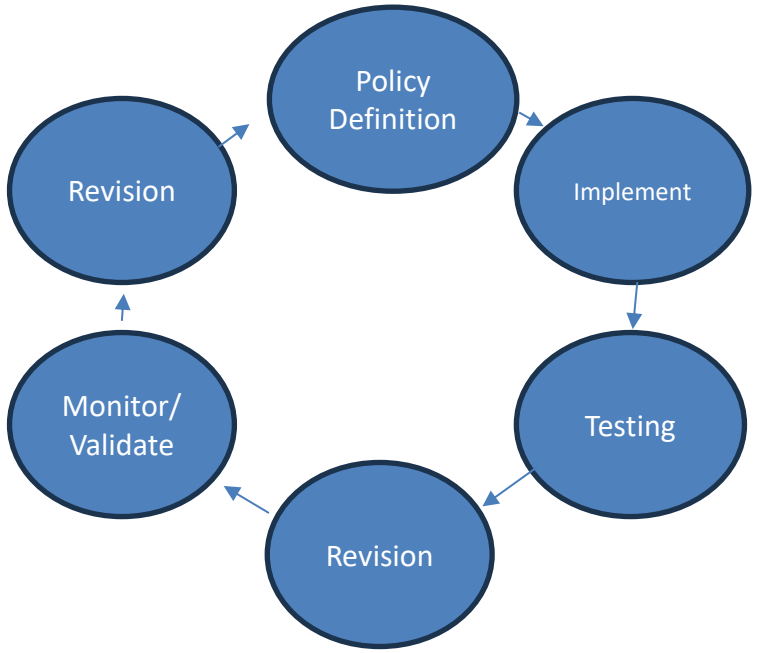


- **Improved Security Posture:** Continuous integration of security practices reduces risks and improves resilience.
- **Higher Quality Software:** Early detection of issues leads to more stable, reliable software.
- **Faster Time-to-Market:** Automating security allows for rapid deployment without compromising safety.
- **Enhanced Team Efficiency:** Cross-functional collaboration minimizes bottlenecks and empowers teams to work together on security goals.

Introduction to Policy-As-Code

Implementing and continuously evaluating security policies as code within the development process to ensure they remain effective, adaptable, and drive consistent compliance.

What is Policy as Code?



Policy as Code is the practice of defining and managing security and compliance policies as version-controlled code.

By **codifying policies**, organizations ensure that security and compliance requirements are automatically enforced within the development pipeline.

This approach enables **consistency**, **traceability**, and **automation**, making it easier to detect and correct policy violations early

Key principles of Policy as Code

Declarative Policy Definitions: Policies are written in a clear, human-readable format, defining the “what” rather than the “how.”

Embedded in CI/CD: Policies are integrated within the CI/CD pipeline to ensure they are validated at every stage of development.

Automation and Enforcement: Policy violations trigger automated alerts or actions, reducing reliance on manual intervention.

Version Control and Collaboration: Policies are treated as code, allowing teams to review, update, and manage them collaboratively.

Why Policy as Code Matters

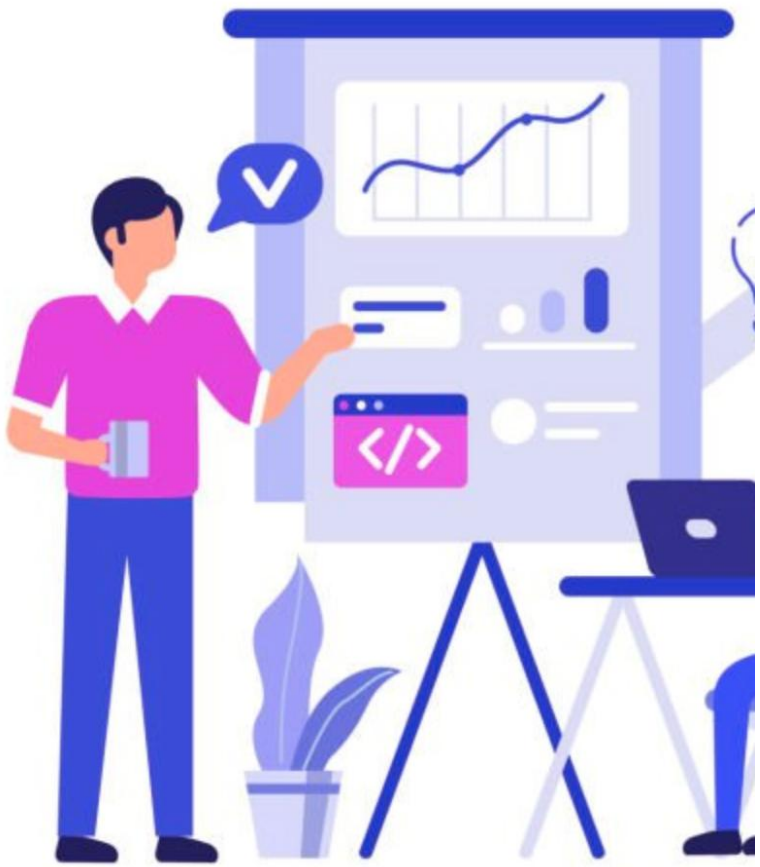
Automation of Compliance: Ensures that compliance checks happen automatically, reducing manual work and human error.

Consistency Across Environments: Policy as Code guarantees that security standards are enforced consistently across all stages and environments.

Adaptability and Agility: Codified policies can be quickly updated to respond to new threats or regulatory changes.

Auditability and Traceability: Version-controlled policies provide an auditable history, helping teams track policy changes over time.

Benefits of Policy as Code



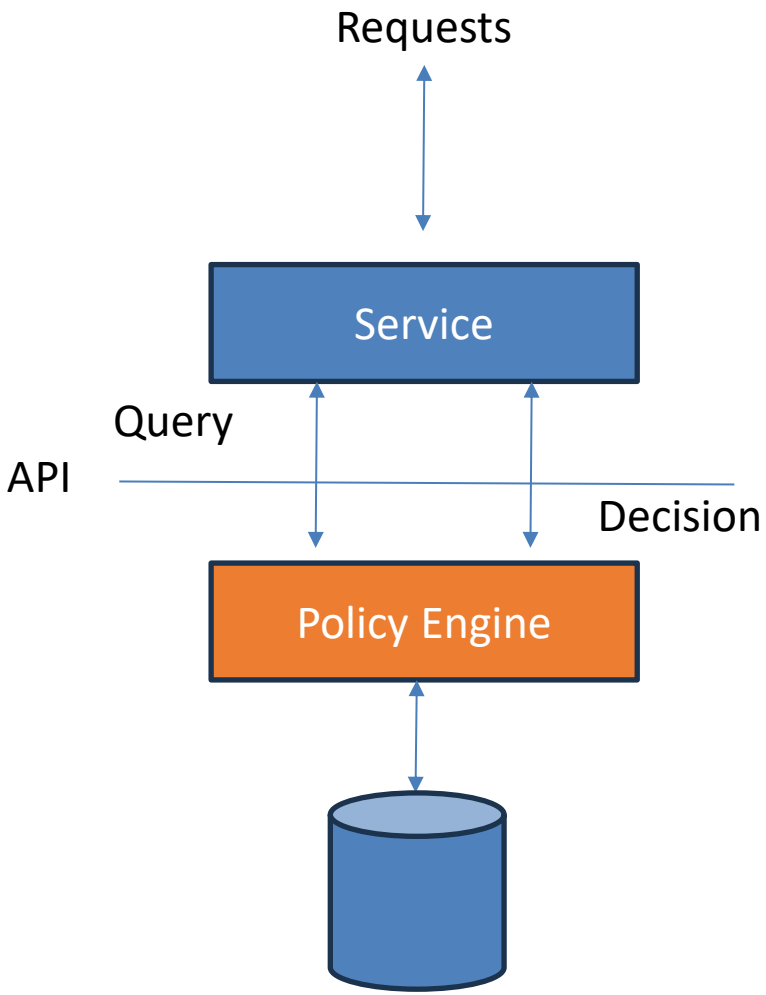
Enhanced Security Posture: By enforcing policies automatically, teams can detect and resolve non-compliance issues sooner.

Reduced Operational Overhead: Automation reduces the need for repetitive manual compliance checks.

Greater Transparency and Accountability: Codified policies provide a clear record of compliance standards and enforcement.

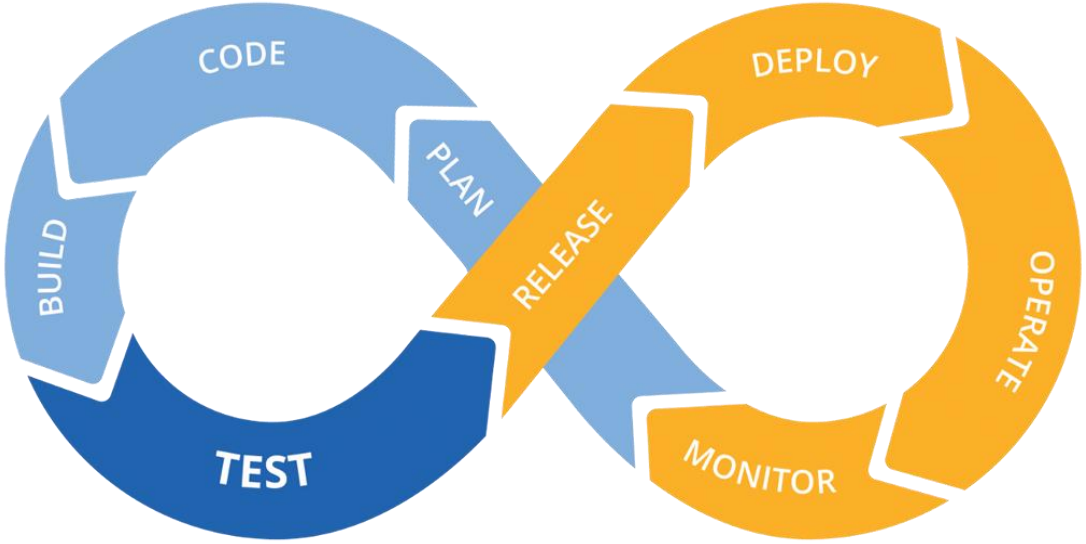
Improved Adaptability: Policies can be updated as code, allowing organizations to quickly respond to new security requirements.

Policy as Code Components



Policy Engine: Interprets policies and returns decisions.
Service: Requests policy checks for specific actions or access.
Requests: Data sent to the policy engine for evaluation.
API: Connects services to the policy engine for seamless policy checks.
Query: The policy engine evaluates input data against policies.
Decision: The result returned to the service, enabling rule enforcement.

Security Integration in the CI/CD Pipeline



Continuously embedding and assessing security checks throughout the CI/CD pipeline to uphold robust, adaptable safeguards and maintain consistent compliance from development to deployment.

Introduction to Security in CI/CD

Integrating Security Throughout: Embedding security at every stage of the CI/CD pipeline enables teams to catch and address vulnerabilities as they arise, preventing issues from reaching production.

Ensuring Compliance and Readiness: By weaving security checks into the pipeline, code remains secure, compliant, and ready for production with each deployment.

Why Integrate Security in CI/CD?

Proactive Threat Mitigation: Identifies vulnerabilities as soon as code is committed.

Cost-Effective: Fixing issues early reduces the cost and complexity of remediation.

Continuous Protection: Ensures secure code delivery by integrating automated security checks.

Key Security Stages in the CI/CD Pipeline

1. **Code Scanning:** Analyzes source code for vulnerabilities.
2. **Dependency Checks:** Validates that third-party libraries are free from known vulnerabilities.
3. **Container Security:** Scans container images for security risks.
4. **Infrastructure as Code (IaC) Scanning:** Checks for misconfigurations in IaC templates.

Benefits of Security Integration in CI/CD

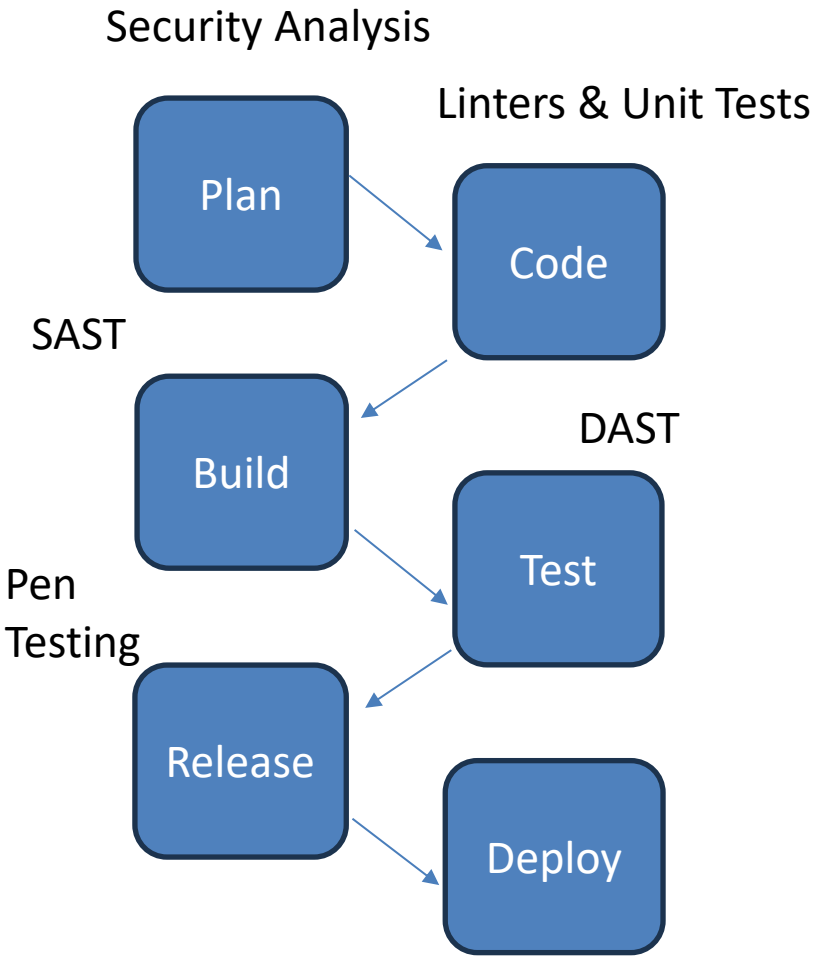
Early Detection: Vulnerabilities are identified before reaching production.

Automated Compliance: Security policies and compliance are enforced automatically.

Enhanced Team Efficiency: Developers receive instant feedback, allowing faster and safer code iterations.

Reduced Risk: Minimizes exposure to potential threats by ensuring secure code deployment.

Overview of CI/CD Security Integrations



Code Scanning: Identifies vulnerabilities directly in the source code before merging.

Dependency Scanning: Checks third-party libraries for known vulnerabilities to ensure secure dependencies.

Container Security: Scans container images for configuration issues and vulnerabilities.

Infrastructure as Code (IaC) Scanning: Validates cloud and infrastructure templates to catch misconfigurations early.

Dynamic Application Security Testing (DAST): Tests the running application for security weaknesses from an attacker's perspective.

GitHub Actions for Code Scanning

```yaml
name: Code Scanning

on:
  pull_request:
    branches: [main]

jobs:
  code-scan:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2

      - name: Run CodeQL Analysis
        uses: github/codeql-action/analyze@v1
        with:
          category: "security"
```

What It Does: Automatically scans the code with CodeQL whenever a pull request is opened to catch vulnerabilities early.

Why It's Useful: Gives developers immediate feedback on security issues, helping to fix them before they reach the main branch.

Impact on Security: Maintains a secure codebase by proactively identifying and resolving security flaws in the source code.

Dependency Scanning with GitHub Actions

```yaml
name: Dependency Scan

on:
  schedule:
    - cron: '0 2 * * 1'

jobs:
  dependabot:
    runs-on: ubuntu-latest
    steps:
      - name: Dependency Scan
        uses: dependabot/fetch-metadata@v1
```

What It Does: Regularly scans all third-party libraries and dependencies to spot vulnerabilities.

Why It's Useful: Ensures that all dependencies remain secure by flagging outdated or insecure libraries.

Reduces risks associated with third-party code, keeping the project safer with up-to-date dependencies.

GitHub Actions for Container Security

```yaml
name: Container Security

on:
  push:
    branches: [main]

jobs:
  container-scan:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2

      - name: Scan Docker Image
        uses: anchore/scan-action@v2
        with:
          image: 'my-image:latest'
```

What It Does: Scans Docker images to identify vulnerabilities in container layers, configurations, and dependencies.

Why It's Useful: Detects security risks within containers before they reach production, ensuring containers are safe to deploy.

Impact on Security: Prevents compromised or misconfigured containers from entering the deployment pipeline, securing the application environment.

IaC Security Checks with GitHub Actions

```yaml
name: IaC Security Scan

on:
  pull_request:
    branches: [main]

jobs:
  iac-scan:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2

      - name: Run tfsec
        uses: aquasecurity/tfsec-action@v1
```

What It Does: Analyzes Infrastructure as Code (IaC) templates to check for misconfigurations or potential security issues.

Why It's Useful: Identifies insecure configurations before deploying infrastructure, helping to secure cloud and on-premises resources.

- Ensures that infrastructure is set up securely from the start, reducing the risk of exposure from insecure configurations.

Automated Security Testing Workflow Example

```yaml
name: Security Workflow

on: [push, pull_request]

jobs:
  security-checks:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2

      - name: CodeQL Analysis
        uses: github/codeql-action/analyze@v1

      - name: Dependency Scan
        uses: dependabot/fetch-metadata@v1

      - name: IaC Scan
        uses: aquasecurity/tfsec-action@v1
```

What It Does: Runs code scanning, dependency checks, and IaC scans in a single, streamlined workflow.

Why It's Useful: Simplifies security by consolidating multiple checks, making it easier to manage and monitor security in one place.

Provides comprehensive security coverage throughout the pipeline, reducing the risk of missing critical vulnerabilities.

Dynamic Application Security Testing (DAST)

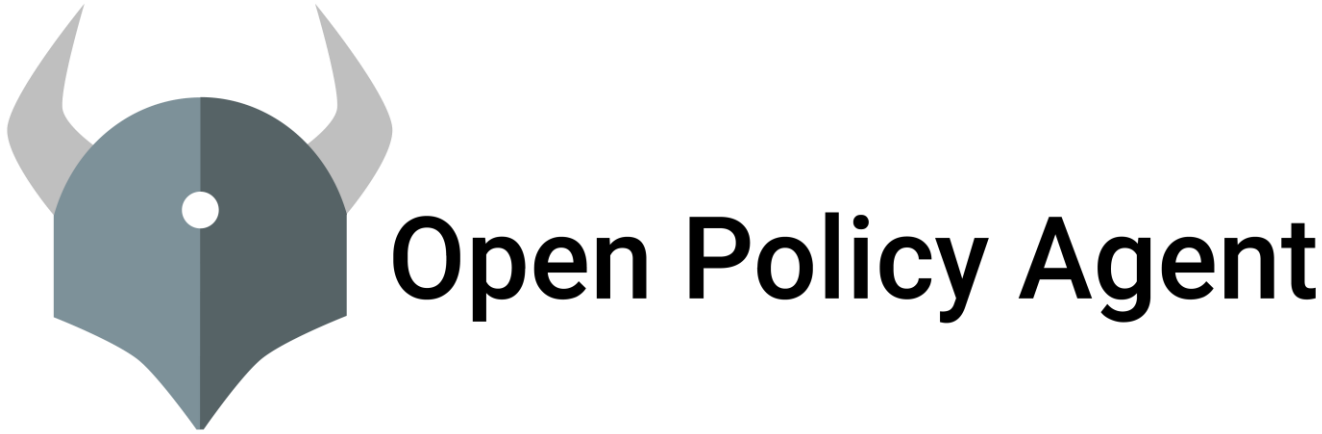
```yaml
name: DAST Security Testing

on:
  push:
    branches: [main]

jobs:
  dast-scan:
    runs-on: ubuntu-latest
    steps:
      - name: Start Application
        run: |
          docker-compose up -d

      - name: OWASP ZAP Scan
        uses: zaproxy/action-full-scan@v0.4.0
        with:
          target: 'http://localhost:8080'
          cmdOptions: '-config api.disablekey=true'

      - name: Stop Application
        run: |
          docker-compose down
```

What It Does: Runs a dynamic security scan against a live instance of the application, identifying vulnerabilities in how it behaves under simulated attack conditions.

Why It's Useful: Tests the application as an attacker would, helping identify security issues that may not be evident in the code alone.

Impact on Security: Provides a final layer of defense by catching vulnerabilities from an external perspective, enhancing overall application resilience.

Introduction to Open Policy Agent (OPA)



Assessing and enforcing policies across systems to maintain effective, adaptable security and compliance throughout the development lifecycle.

What is Open Policy Agent (OPA)

Open Policy Agent

OPA is an open-source, general-purpose policy engine that unifies policy enforcement across different systems.

- It enables organizations to define and enforce policies consistently across applications, microservices, Kubernetes, CI/CD pipelines, and more.
- By centralizing policy management, OPA makes it easier to update, audit, and apply security and compliance policies.

Key Benefits of OPA

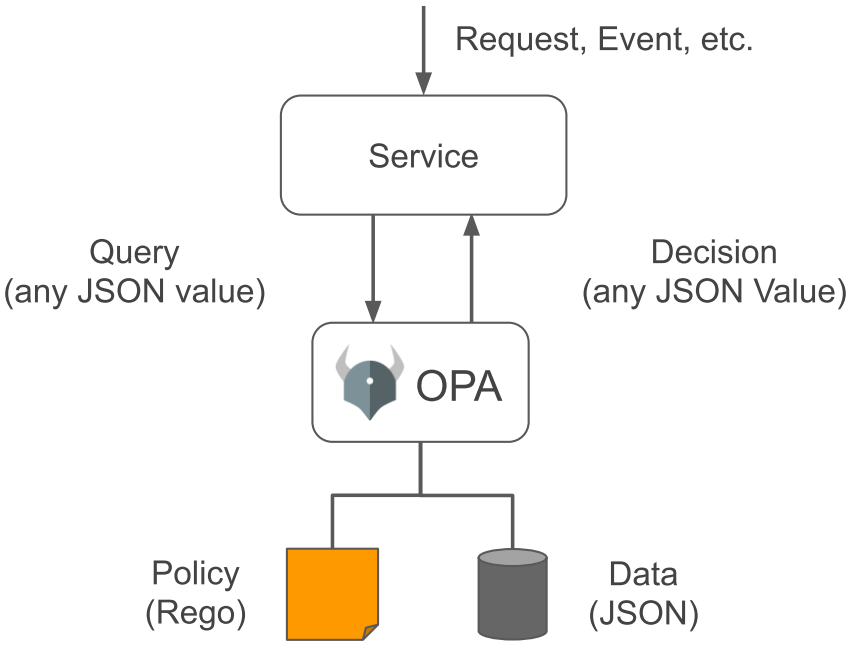
Open Policy Agent

Consistency: Standardizes policy enforcement across various environments and applications.

Flexibility: Allows policies to be written in a declarative language, Rego, adaptable to different use cases.

Auditability: Policies are version-controlled and auditable, supporting compliance requirements.

OPA Architecture Overview



Policy Engine: The OPA engine interprets policies written in Rego and makes real-time policy decisions.

Data Input: OPA evaluates policies against structured data (JSON) provided by applications or services.

Policy Decision API: Applications call OPA's API to receive authorization or policy decisions.

Writing a Basic OPA Policy

```rego
package authz

allow {
  input.user == "admin"
}
```

This example shows a simple policy that checks if a user has access to a resource.

The policy allows access if the user in the input data is “admin.”

Applications can call OPA with JSON input like {"user": "admin"}, and OPA returns whether access is allowed.

Policy Enforcement in Microservices

```rego
package authz

default allow = false

allow {
  input.user.role == "manager"
}
```

Enforce policies directly within microservices, ensuring consistent rules across the application.

- In this example, access is allowed only if the user's role is "manager."
- This simplifies role-based access control in microservices by centralizing the logic in OPA.

Kubernetes Admission Control with OPA

```rego
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == "Pod"
  input.request.object.spec.containers[_]
    .securityContext.runAsUser == 0
  msg = "Running as root is not allowed"
}
```

OPA can be used to enforce security policies on Kubernetes clusters, such as blocking deployments with specific configurations.

- This example policy prevents deployment if a container runs as root, improving security.

Implementing CI/CD Pipeline Policies

```rego
package ci.policy

deny[msg] {
  input.vulnerabilities[_].severity == "critical"
  msg = "Build blocked due to critical vulnerability"
}
```

Enforce policies directly in CI/CD pipelines to ensure compliance before deployment.

- This example policy blocks builds if a critical vulnerability is detected, ensuring safer deployments.

Integrating OPA with REST APIs

```rego
package api.access

allow {
  input.user.type == "employee"
  input.resource.accessible == true
}
```

OPA policies can be tested with unit tests to ensure they work as expected.

- The example tests confirm that only “admin” users are allowed, ensuring policy accuracy.

Testing and Validating OPA Policies

```rego
package authz

test_allow_admin {
  allow with input as {"user": "admin"}
}

test_deny_non_admin {
  not allow with input as {"user": "guest"}
}
```

OPA policies can be tested with unit tests to ensure they work as expected.

- The example tests confirm that only “admin” users are allowed, ensuring policy accuracy.

OPA Policy Evaluation in Action

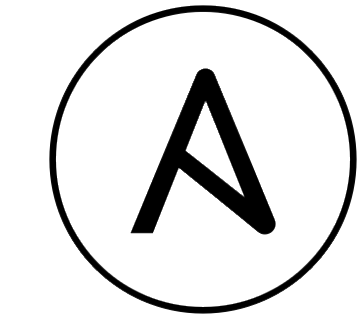
Open Policy Agent

Running OPA Locally: Use OPA run to load policies and data locally, allowing you to test policies before deploying.

Deploying in Production: OPA can be deployed as a sidecar or standalone service, integrating directly into application infrastructure.

OPA Decision Logging: Logs all policy decisions, providing visibility into enforcement for auditing and troubleshooting.

Git

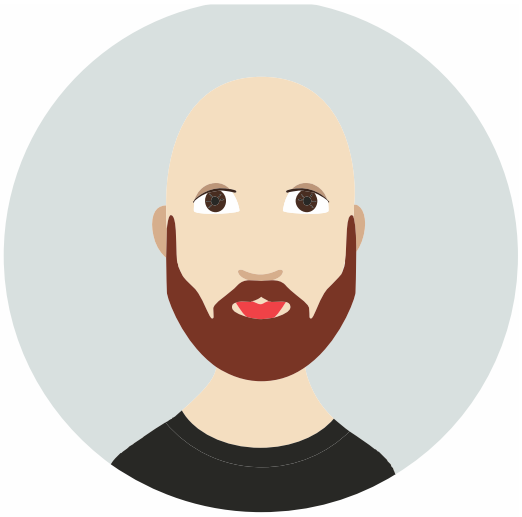


ANSIBLE

HashiCorp Terraform

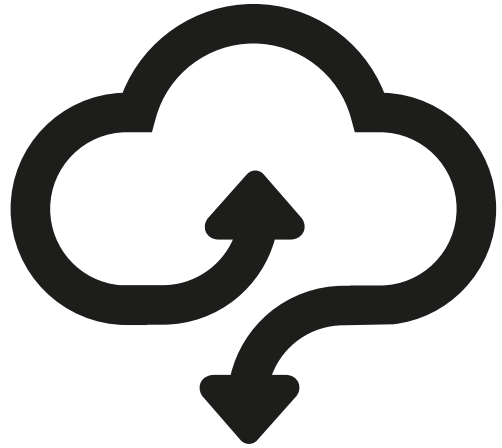
python™

Git



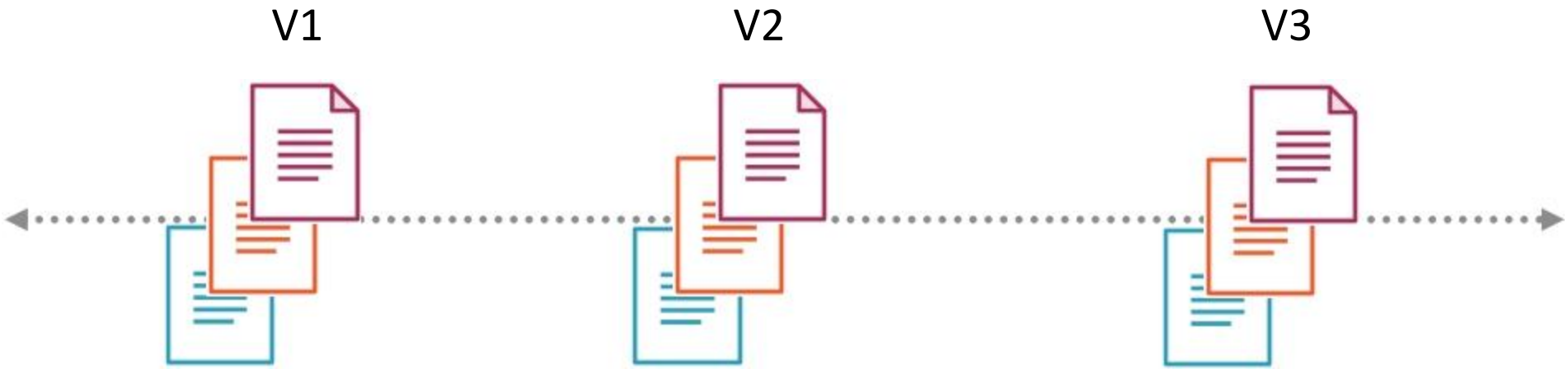
git

Git



What is Git?

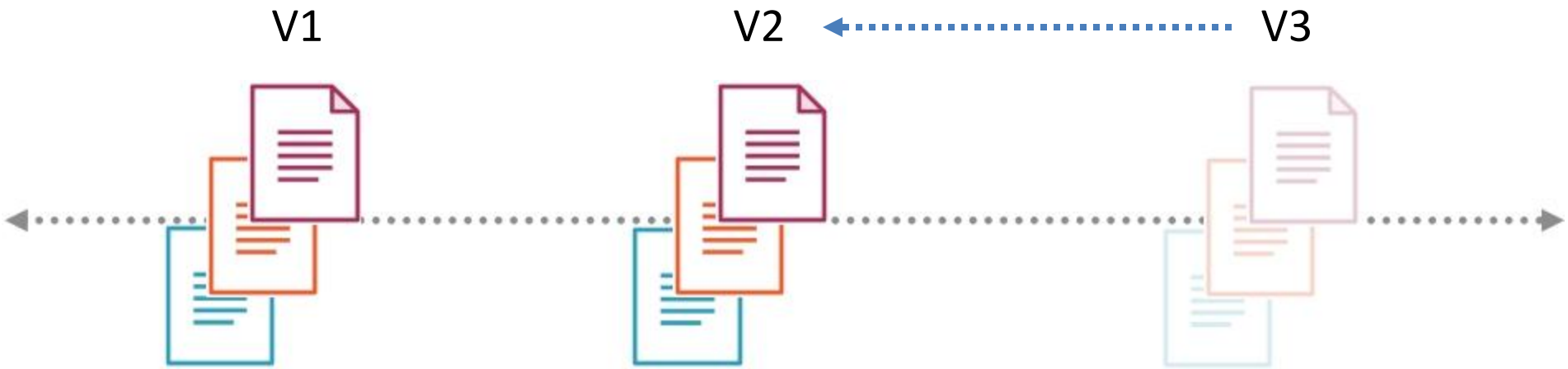
Version Control System



What is Git?

Version Control System

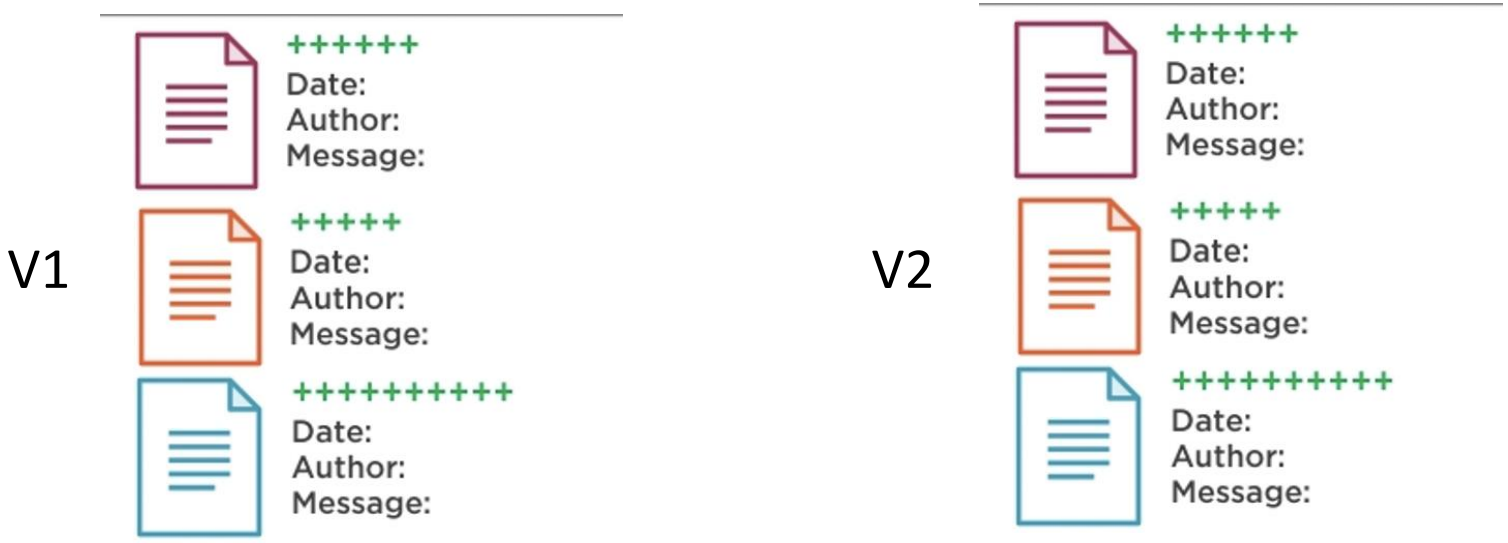
- Software designed to record changes to files made over time
- Ability to revert back to a previous file version or project version



What is Git?

Version Control System

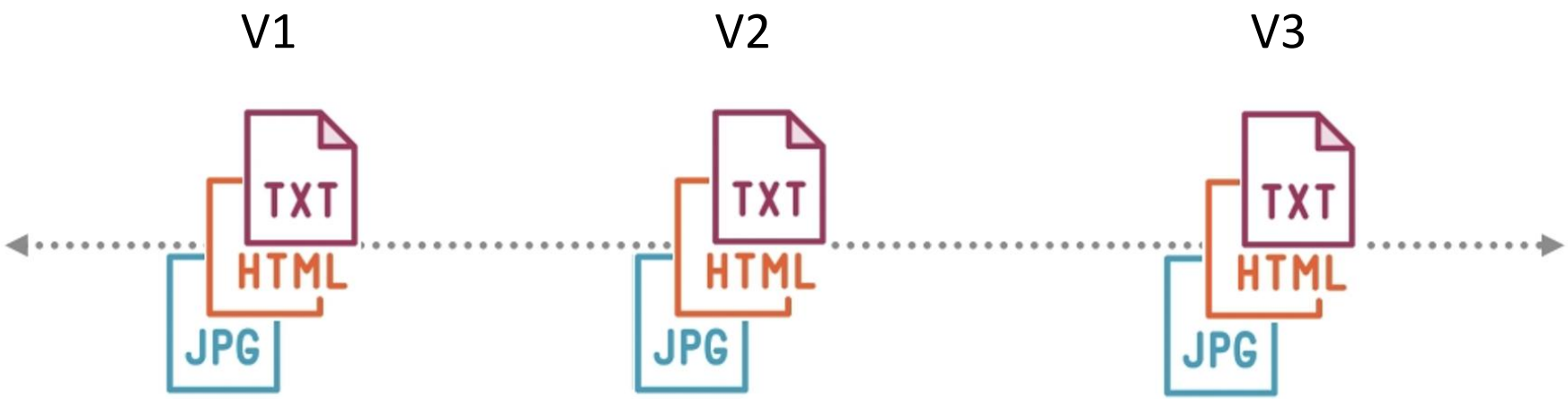
- Software designed to record changes to files made over time
- Ability to revert back to a previous file version or project version
- Compare changes made to files from one version to another



What is Git?

Version Control System

- Software designed to record changes to files made over time.
- Ability to revert back to a previous file version or project version.
- Compare changes made to files from one version to another.
- Ability to version control any plain text file, not just source code.



Git use-case

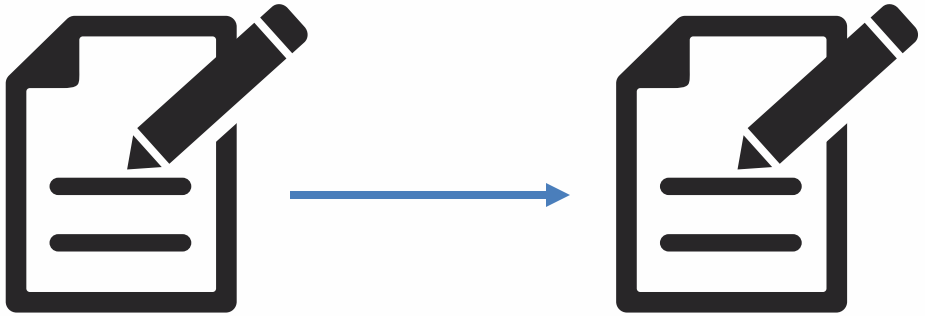
WIRED BRAIN
COFFEE

Git use-case

Latte

Git use-case

Latte



POP QUIZ: Dangers of copying files

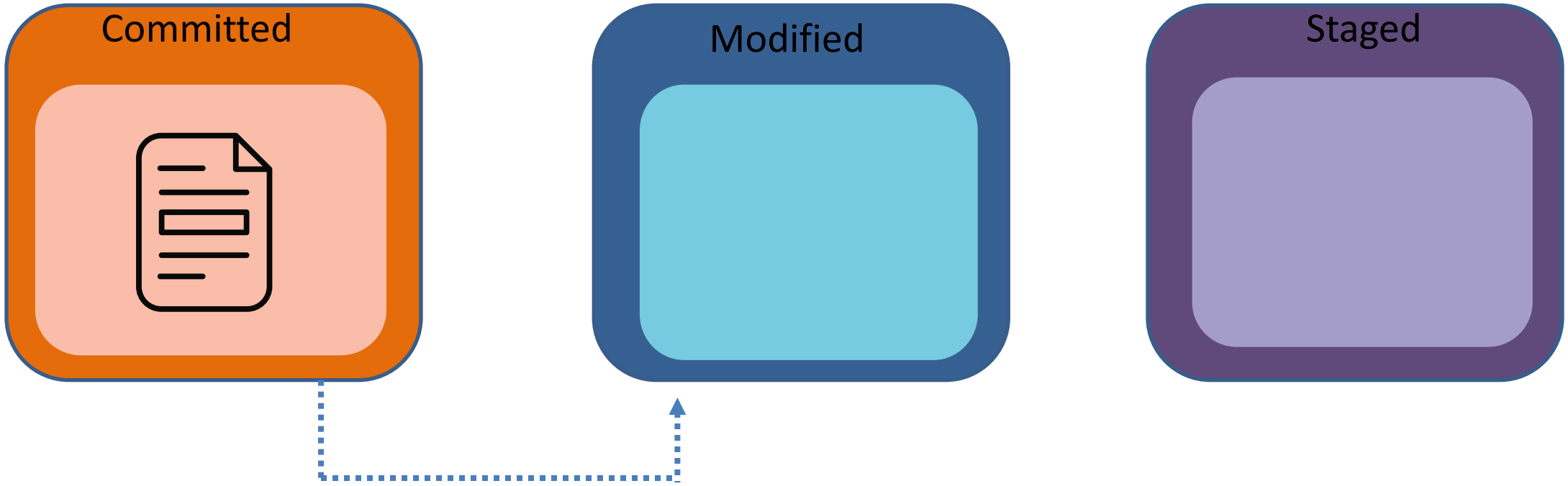
What are some dangers of Steve's current process?

POP QUIZ: Dangers of copying files

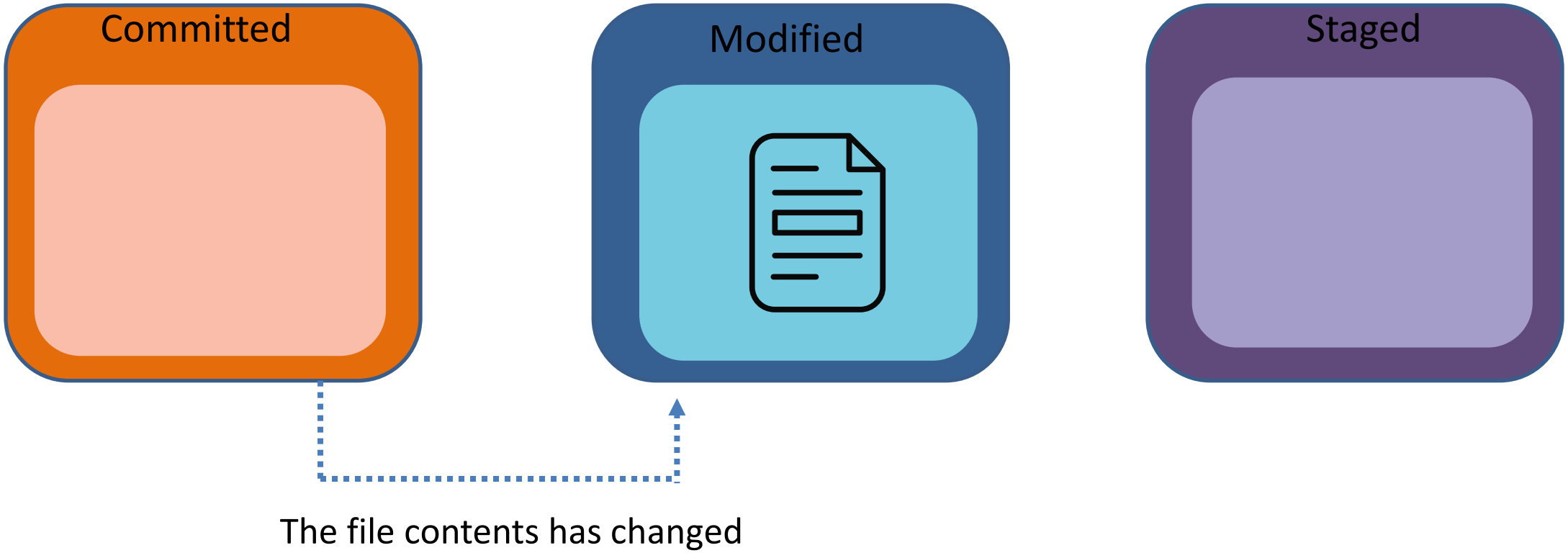
What are some dangers of Steve's current process?

- Files live in one location
- Accidentally overwrite existing recipes

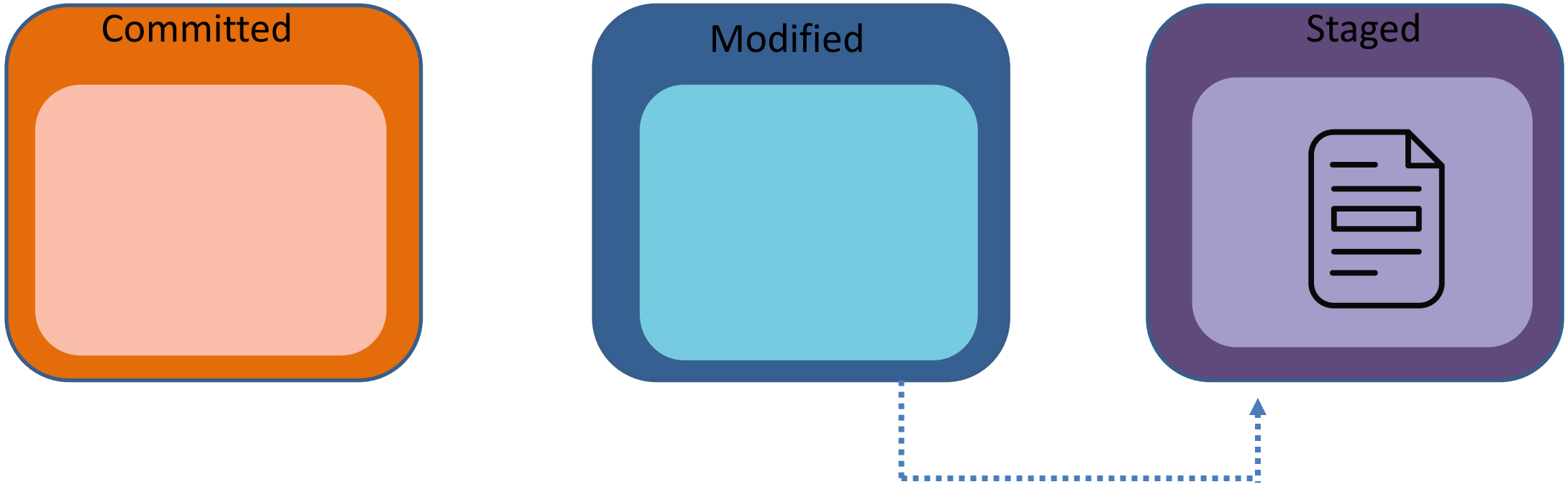
Three stages of a file



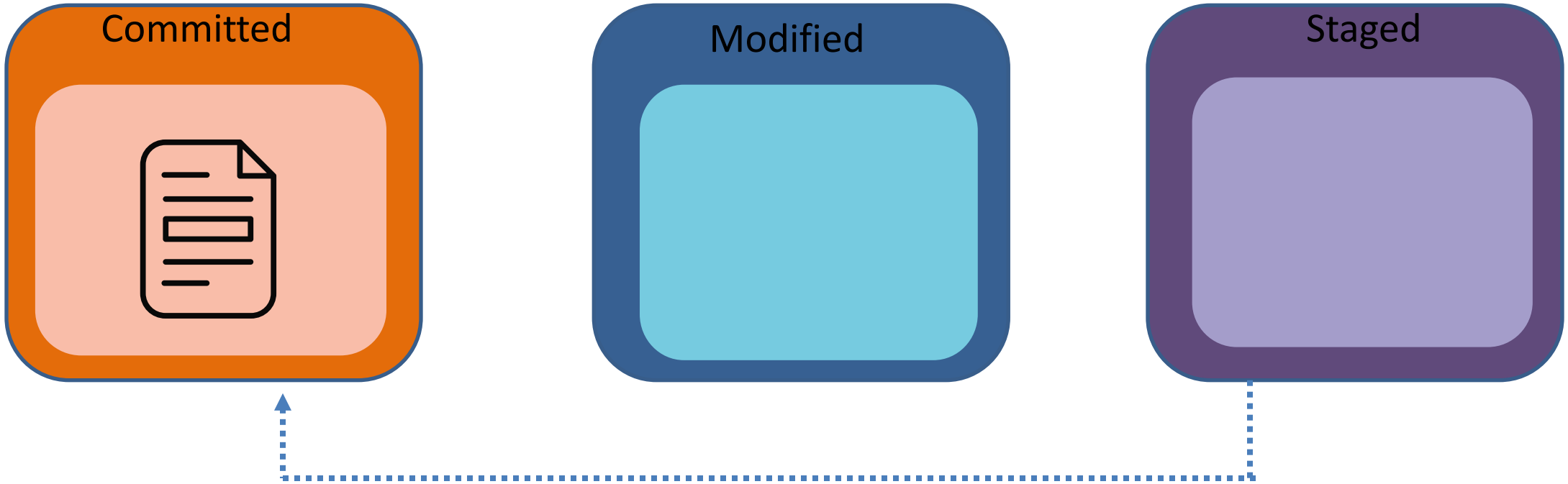
Three stages of a file



Three stages of a file

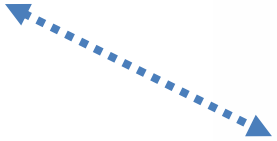


Three stages of a file



Git providers

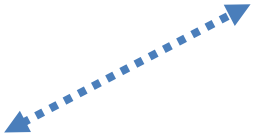
GitHub



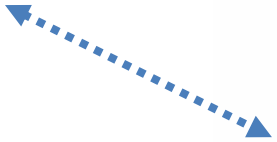
Git providers

GitLab

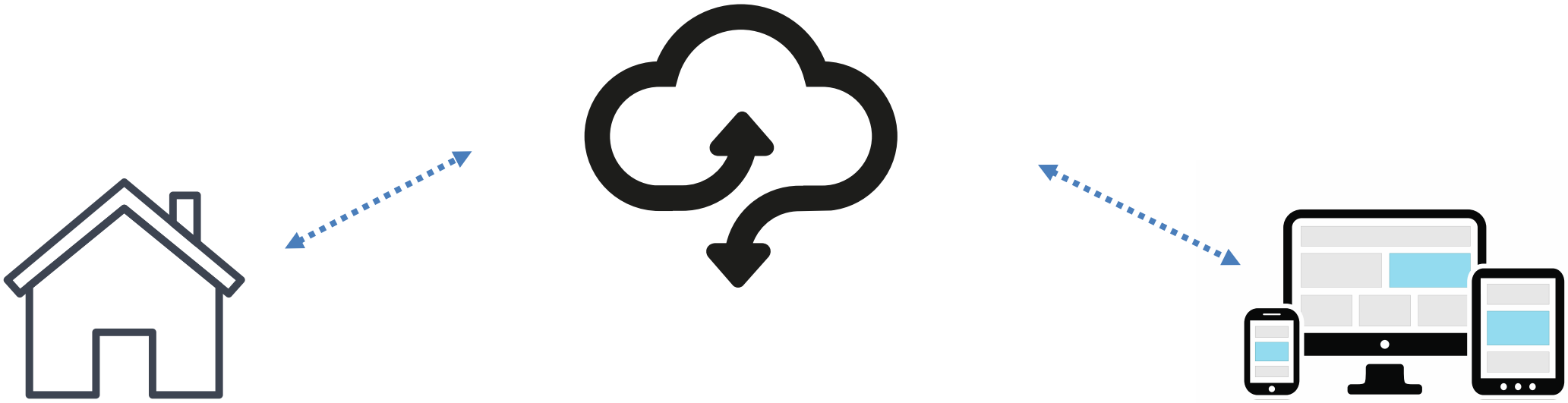
Git providers



Bitbucket



Git providers



Git basic commands

```
Clone from upstream:

    $ git clone git://git.kernel.org/pub/scm/.../linux.git my-linux
    $ cd my-linux
```

- `git clone` is used to copy a repository
  - You can also clone a local repository
  - `git clone /path/to/repository`

Git basic commands

```
Start a new Git repository for an existing code base

        $ cd /path/to/my/codebase
        $ git init      (1)
        $ git add .     (2)
        $ git commit    (3)

    1. Create a /path/to/my/codebase/.git directory.
    2. Add all existing files to the index.
    3. Record the pristine state as the first commit in the history.
```

- Use `git init` to create a new local repository.
- Alternatively, you can create a repository within a new directory by specifying the project name: `git init [project name]`

Git basic commands

```
$ git add git-*.sh
```

- Use `git add <files/directories>` to add files to staging area.

Git basic commands

```
$ git commit
```

- `git commit` will create a snapshot of the changes and save it to the git directory.
- `-m` will pass a commit message: `git commit -m "commit message"`

Git basic commands

```
git config --add core.gitproxy '"proxy-command" for example.com'
```

- `git config` can be used to set user-specific configuration values like email, username, file format, and so on.
- `git config --global user.email youremail@example.com`
- `--global` sets value for all repositories

Git basic commands

```
git status [<options>...] [--] [<pathspec>...]
```

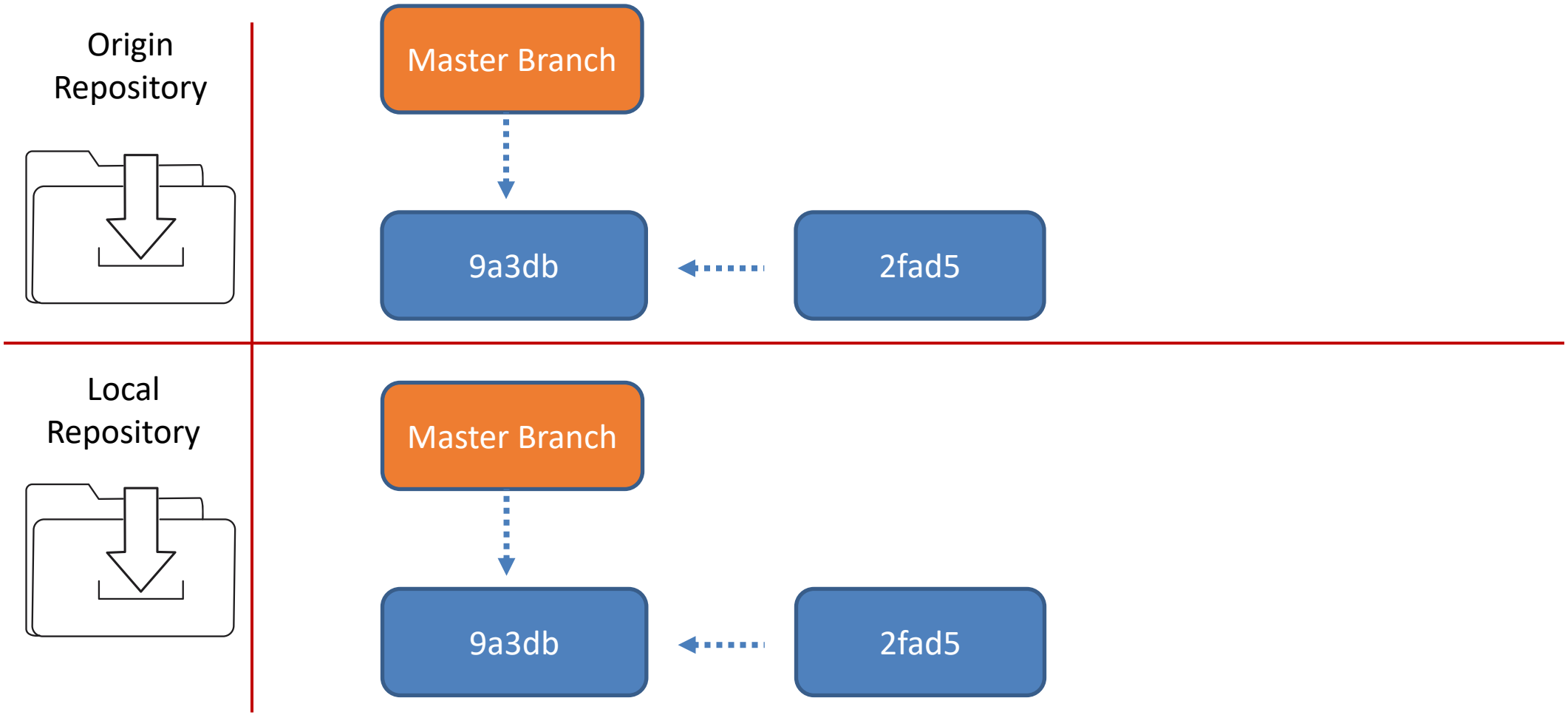
- `git status` displays the list of changed files together with the files that are yet to be staged or committed.

Git basic commands

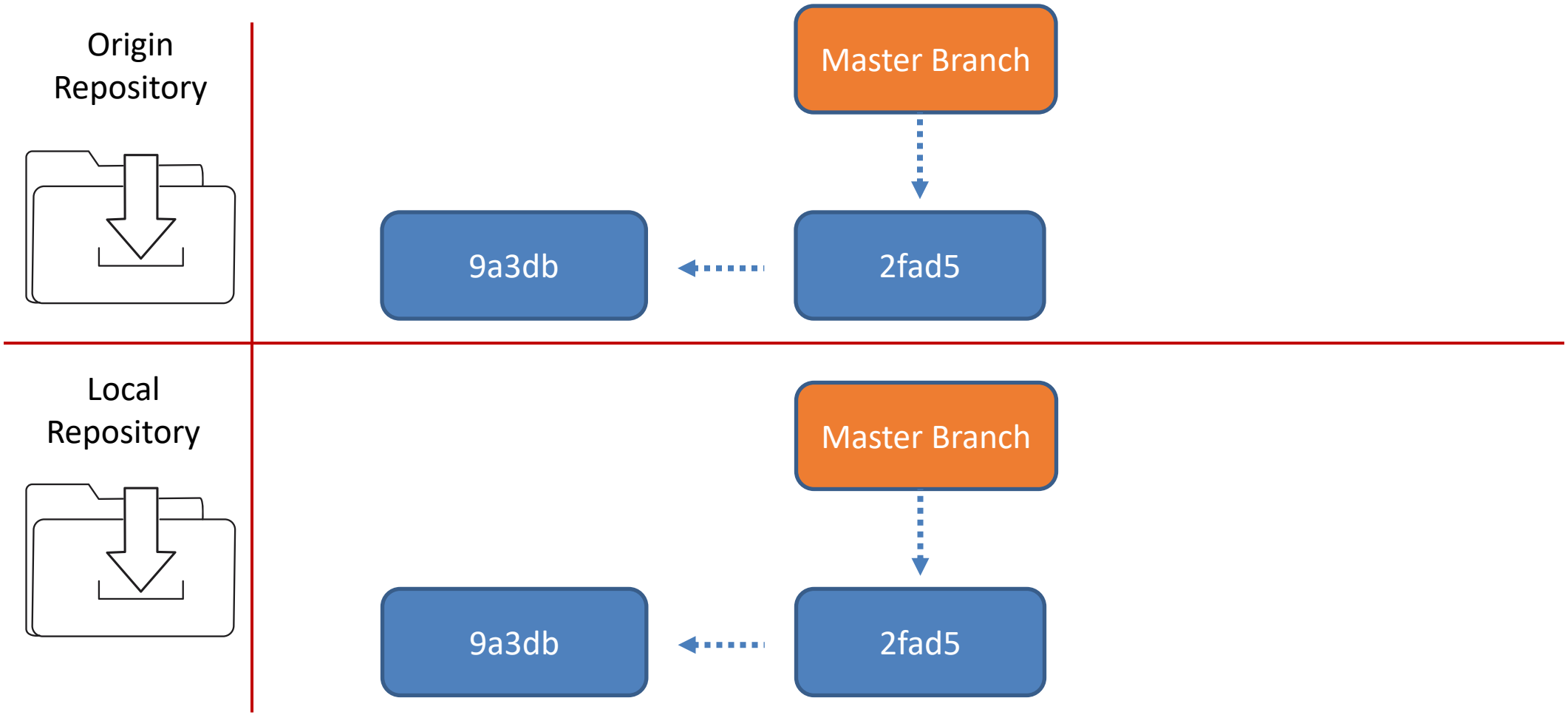
```
git push origin master
```

- `git push` is used to send local commits to the master branch of the remote repository.
- `git push origin <branch>`

Git commit process



Git commit process



Git diff

```
Various ways to check your working tree

        $ git diff              (1)
        $ git diff --cached     (2)
        $ git diff HEAD         (3)
```

- `git diff` lists down conflicts.
- Use it to show changes that have been made but have not been committed.

Git log

```
git log --no-merges
    Show the whole commit history, but skip any merges
```

```
commit a68112e12f896ca028f25a46e58aaf46360cb847
Author: Jason Smith <jason@smithss.org>
Date:   Sun Jan 6 21:12:35 2019 +0000

    initial
```

- `git log` is used to see the repositories history by listing certain commit details.

Git blame

```
git blame CNAME
```

```
a76f1de5 (Jason Smith 2020-04-19 22:52:29 -0700 1) kickstart.innovationin.software
5948ef48 (Jason Smith 2020-04-19 22:51:19 -0700 2)
```

- `git blame` is used to see which users changed a file.

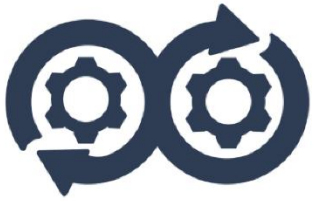
Git blame

```
$ git commit ...
$ git reset --soft HEAD^      (1)
$ edit                        (2)
$ git commit -a -c ORIG_HEAD  (3)
```

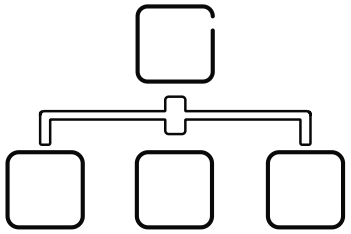
```
Undo commits permanently

        $ git commit ...
        $ git reset --hard HEAD~3   (1)
```

- `git reset` undo changes
- `--soft` `--mixed` or `--hard` depending on situation.
- `--soft` Does NOT change local working directory. Index still points to latest commit
- `--mixed` Does NOT change local working directory. Index points to reverted commit
- `--hard` Changes local working directory and index to point to reverted commit. Deletes all comments and files from original commit.

Lab: Git

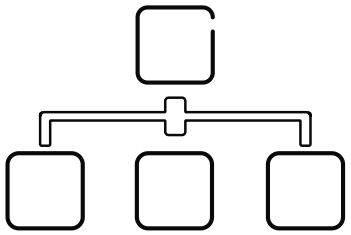


Software Development flows



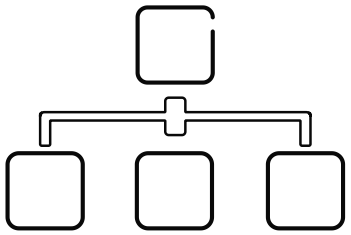
- Centralized flow
- GitHub
- Featured branch

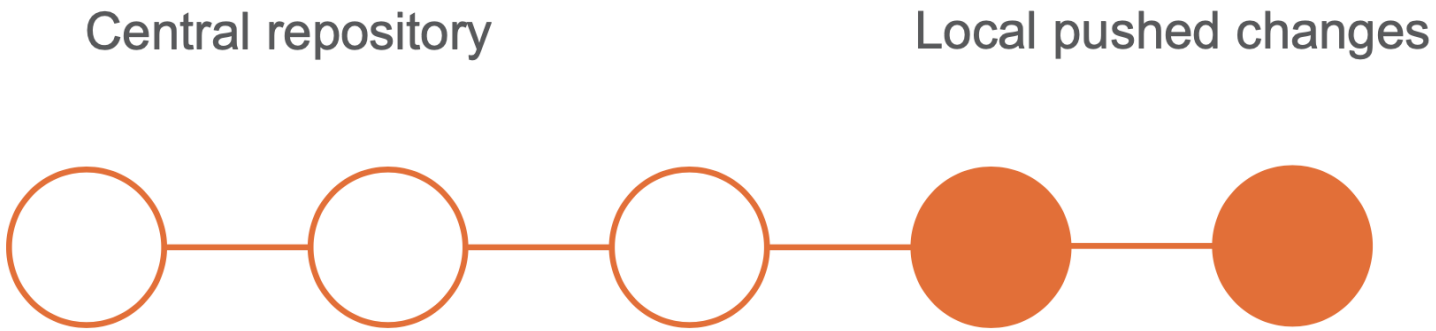
Centralized flow



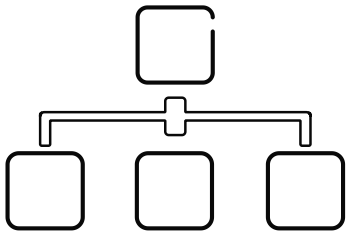
- Uses a centralized repository
- End users 'clone' repository
- Pull changes from central repository to local
- Make changes
- Push local changes back to centralized repository
- Similar to SVN

Centralized flow



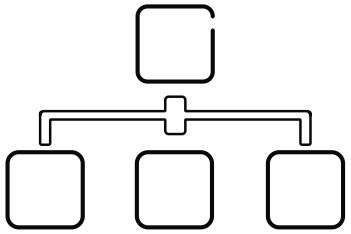


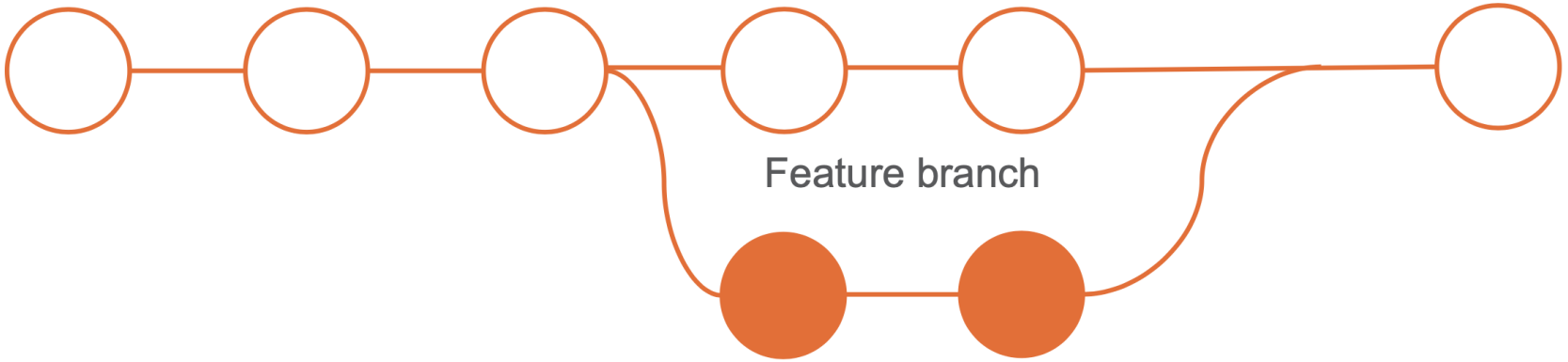
Feature branch flow



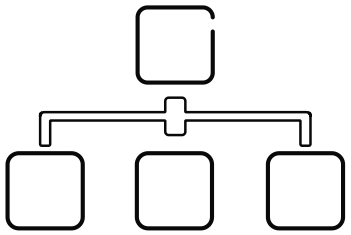
- Create a feature branch
- Add new code
- Open a pull request
- Merge code back into Master

Feature branch flow

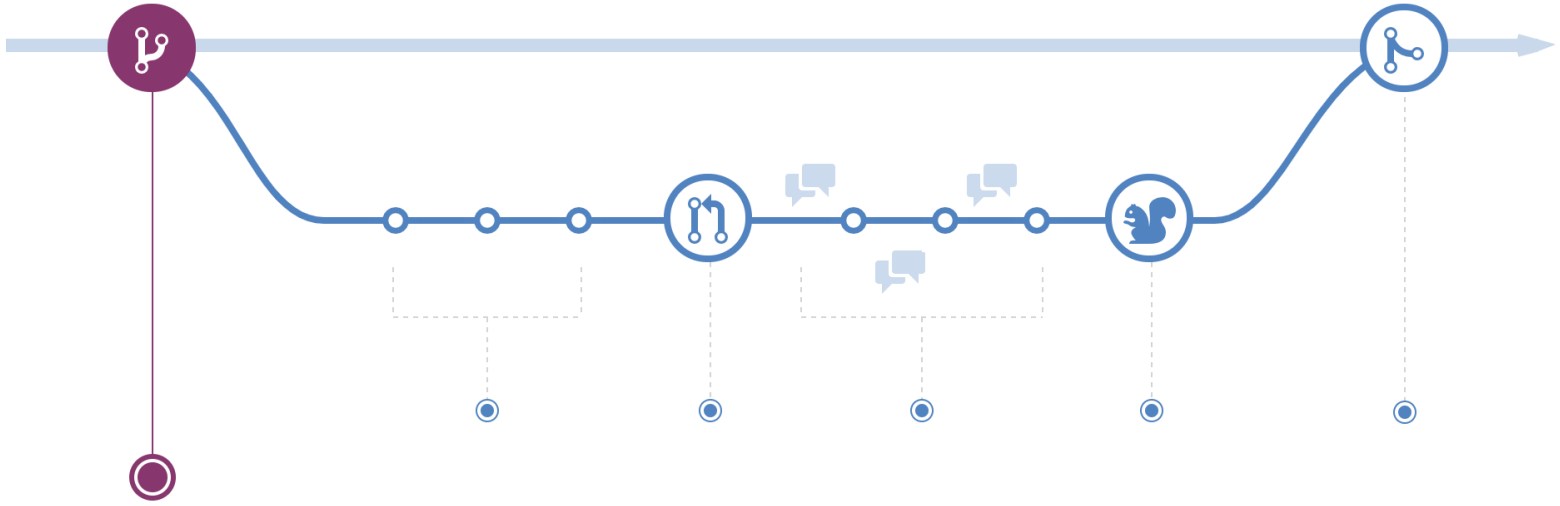




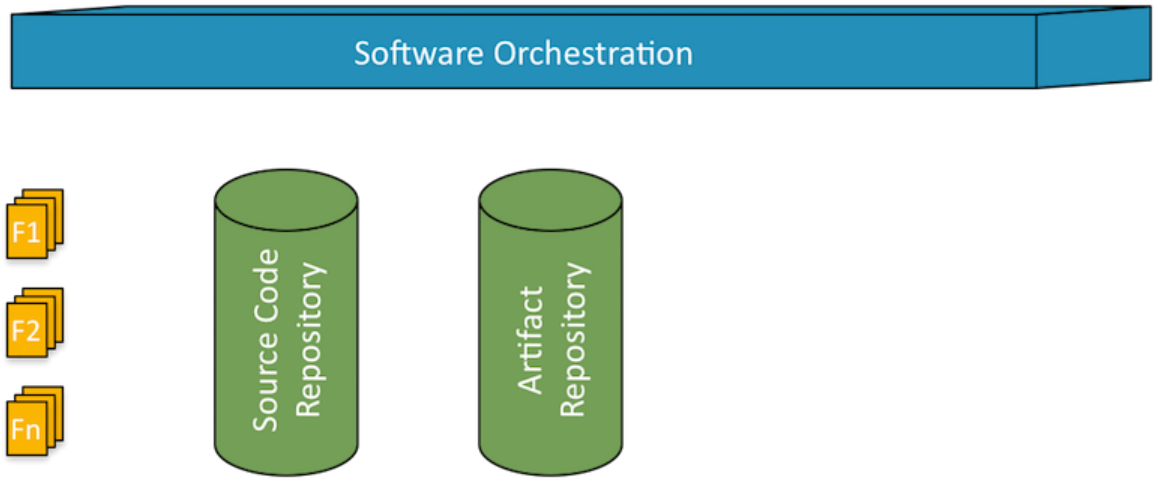
GitHub flow



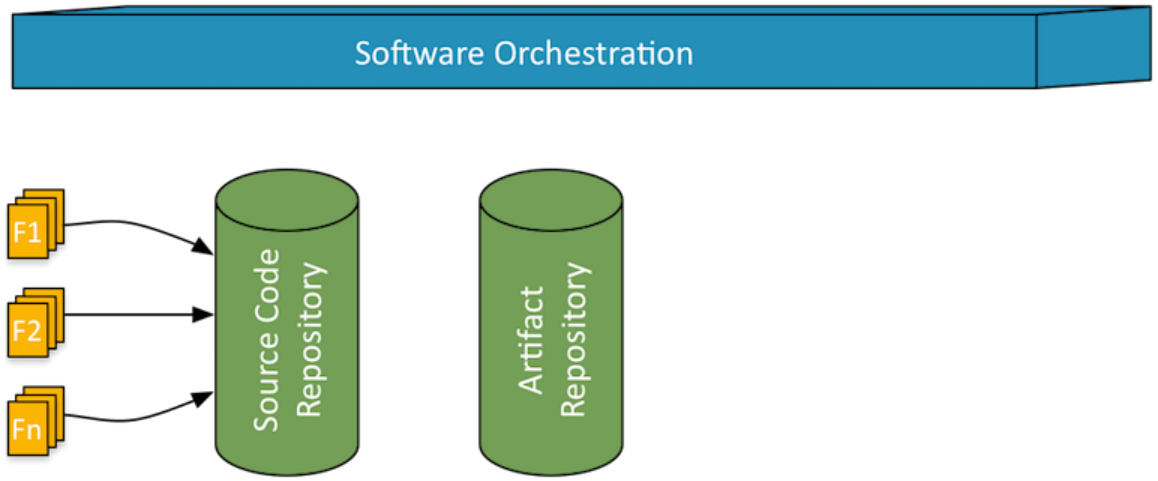
1. Create a branch



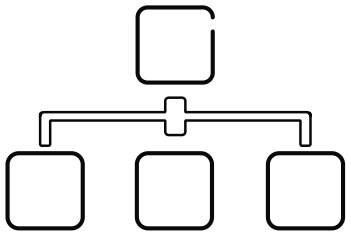
GitHub flow



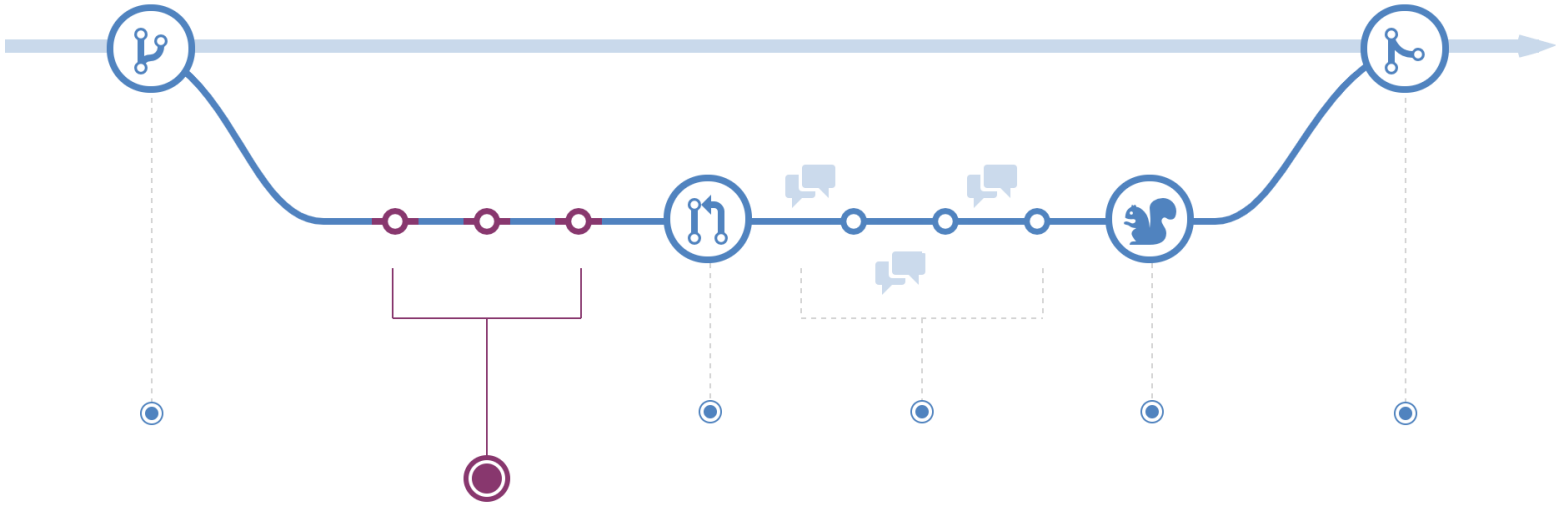
GitHub flow



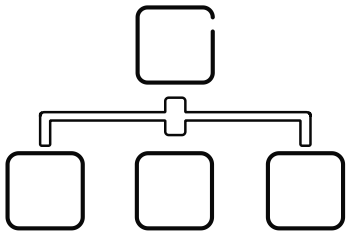
GitHub flow



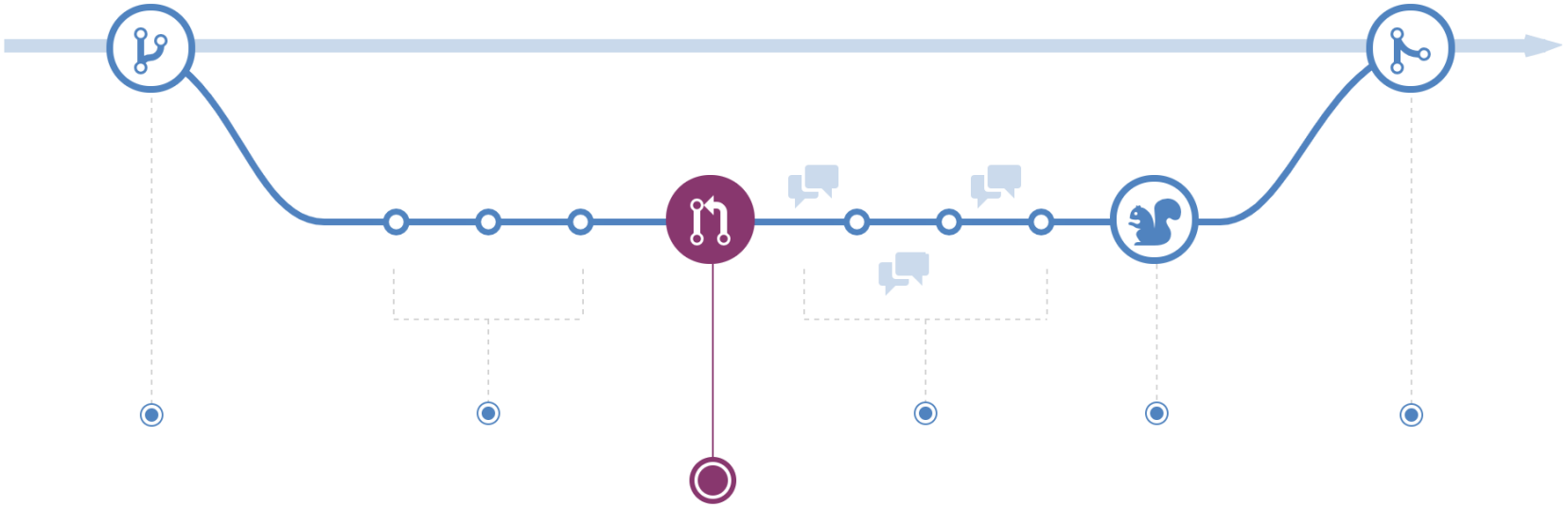
1. Create a branch
2. Checkout branch and make changes



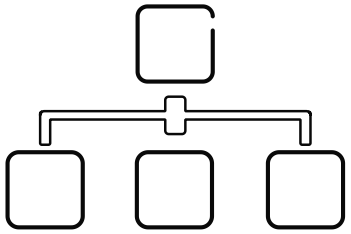
GitHub flow



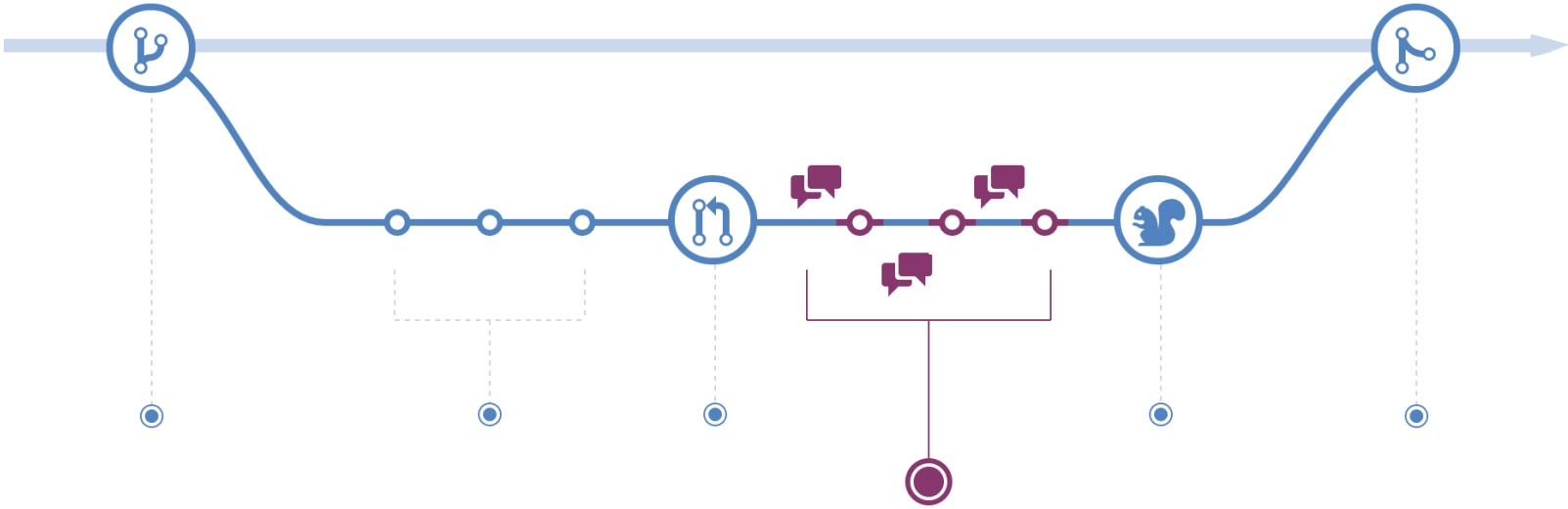
1. Create a branch
2. Checkout branch and make changes
3. Open pull request



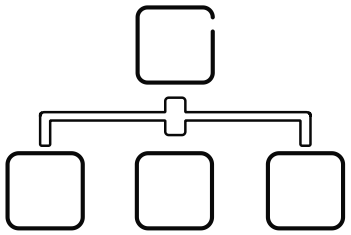
GitHub flow



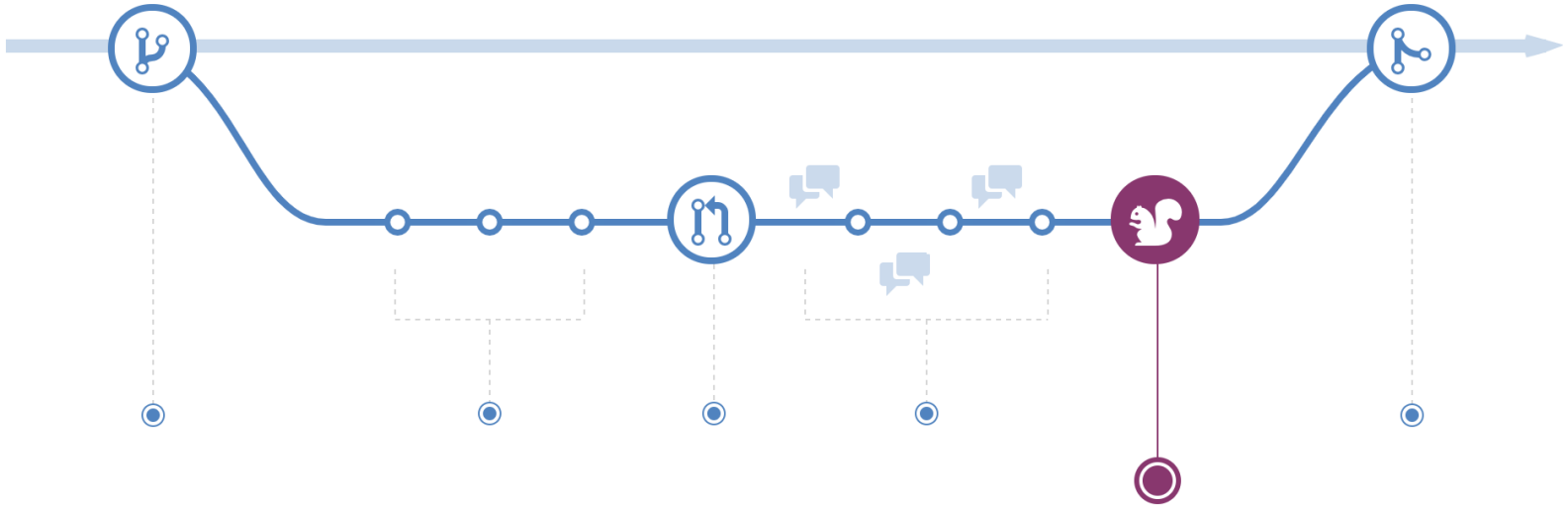
1. Create a branch
2. Checkout branch and make changes
3. Open pull request
4. Peer review



GitHub flow



1. Create a branch
2. Checkout branch and make changes
3. Open pull request
4. Peer review
5. Deploy

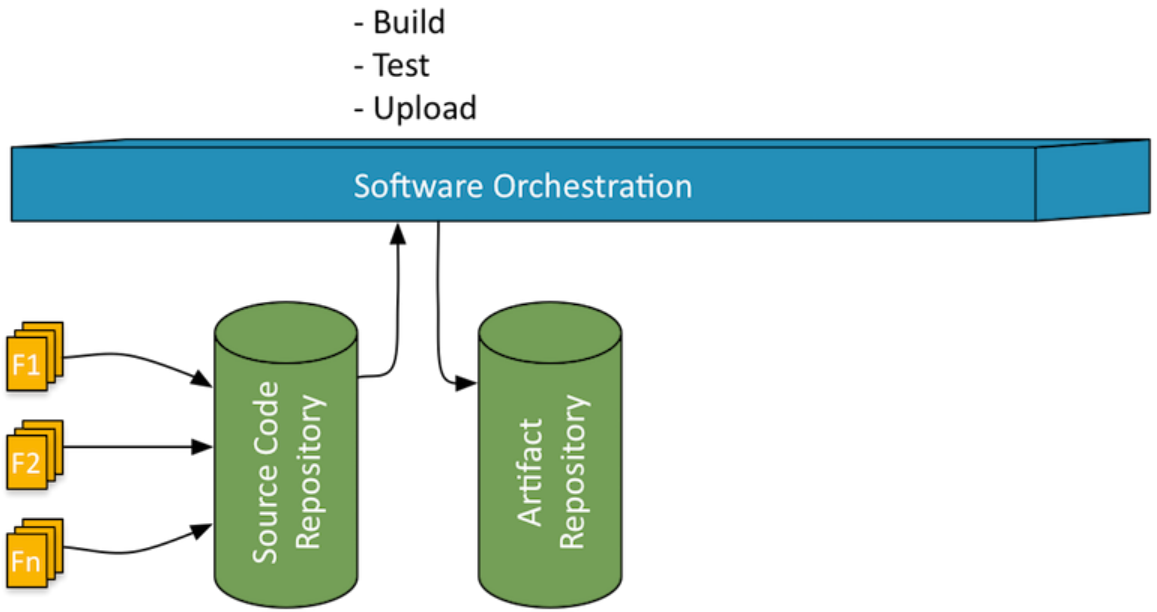


POP QUIZ: Your development flow

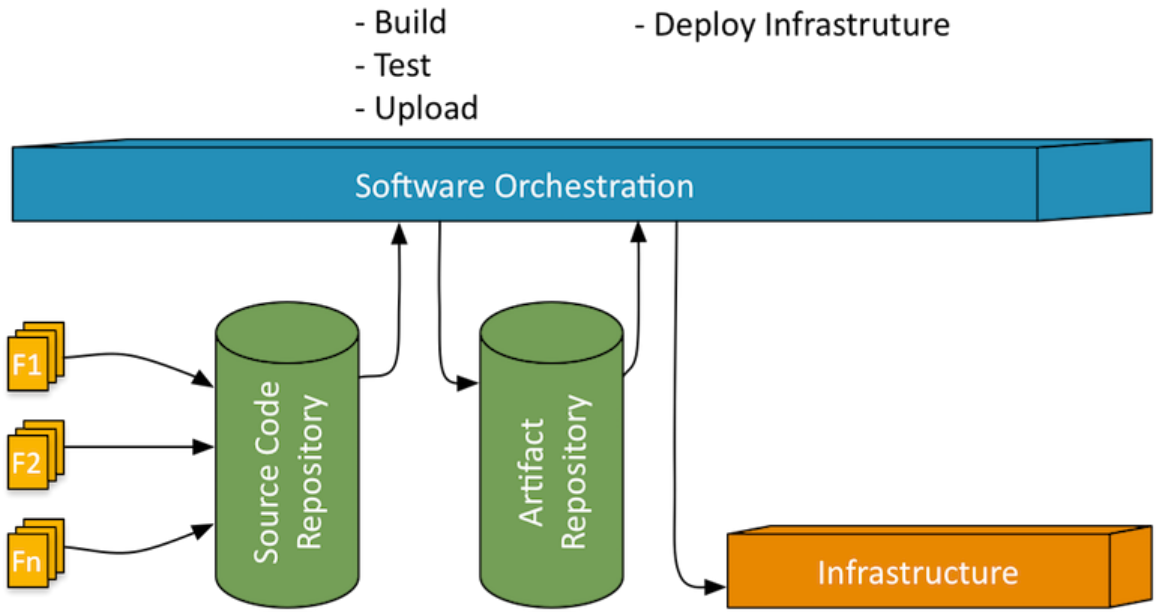
How does your team add new features to your applications?

- Feature branches?
- All changes to master?

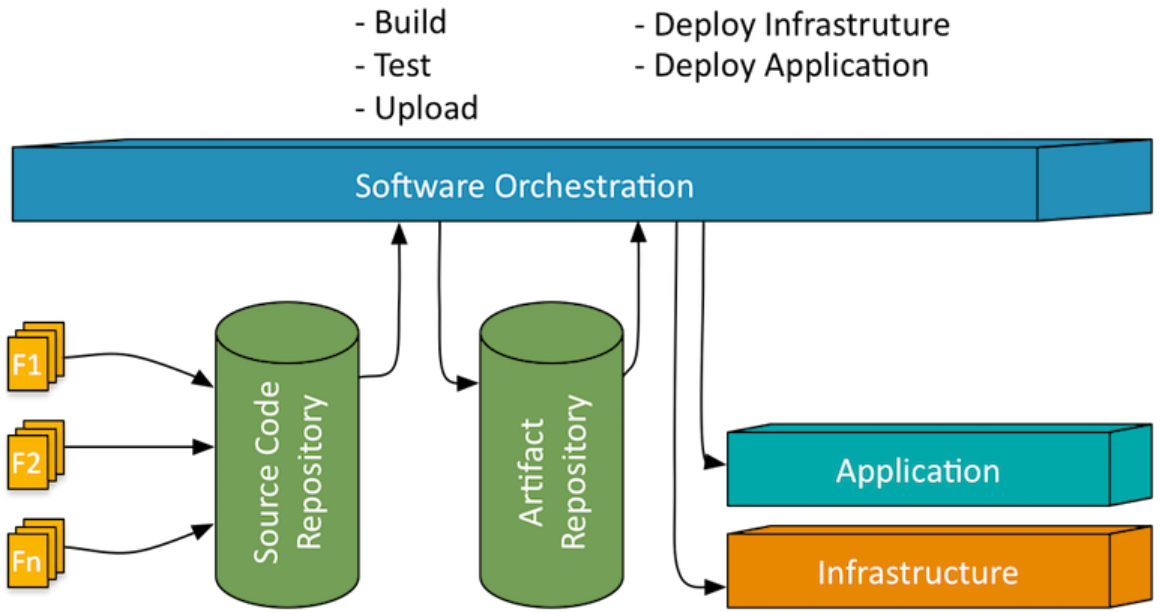
Continuous Integration & Testing



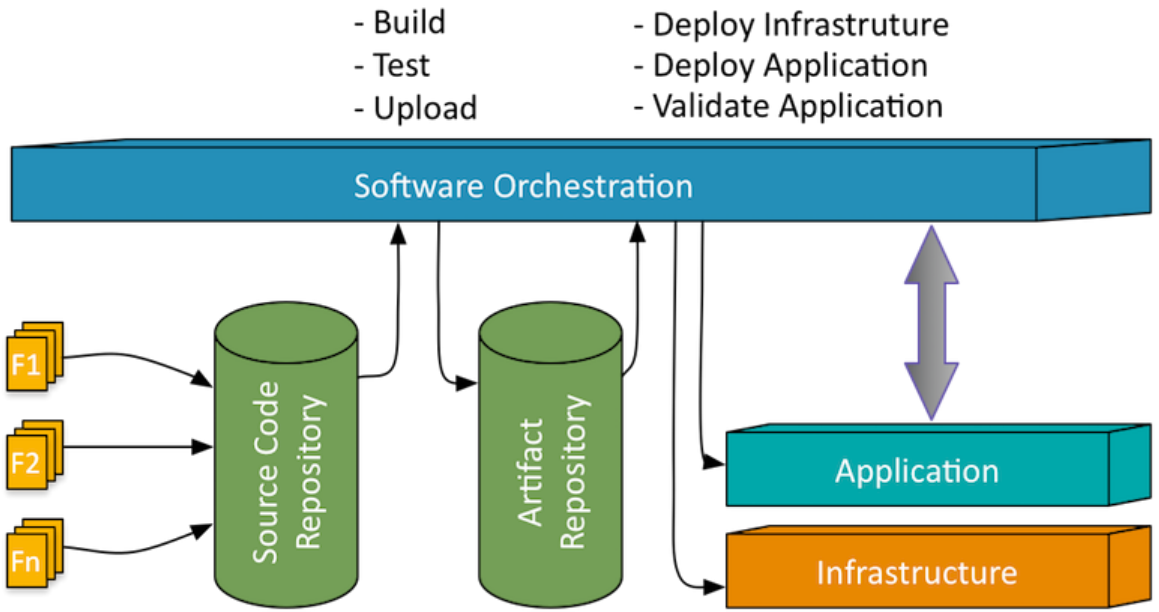
Continuous Release and Deployment



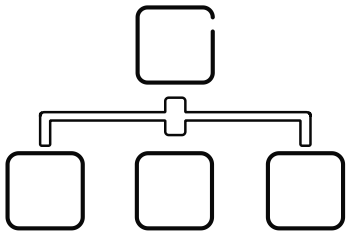
Continuous Release and Deployment



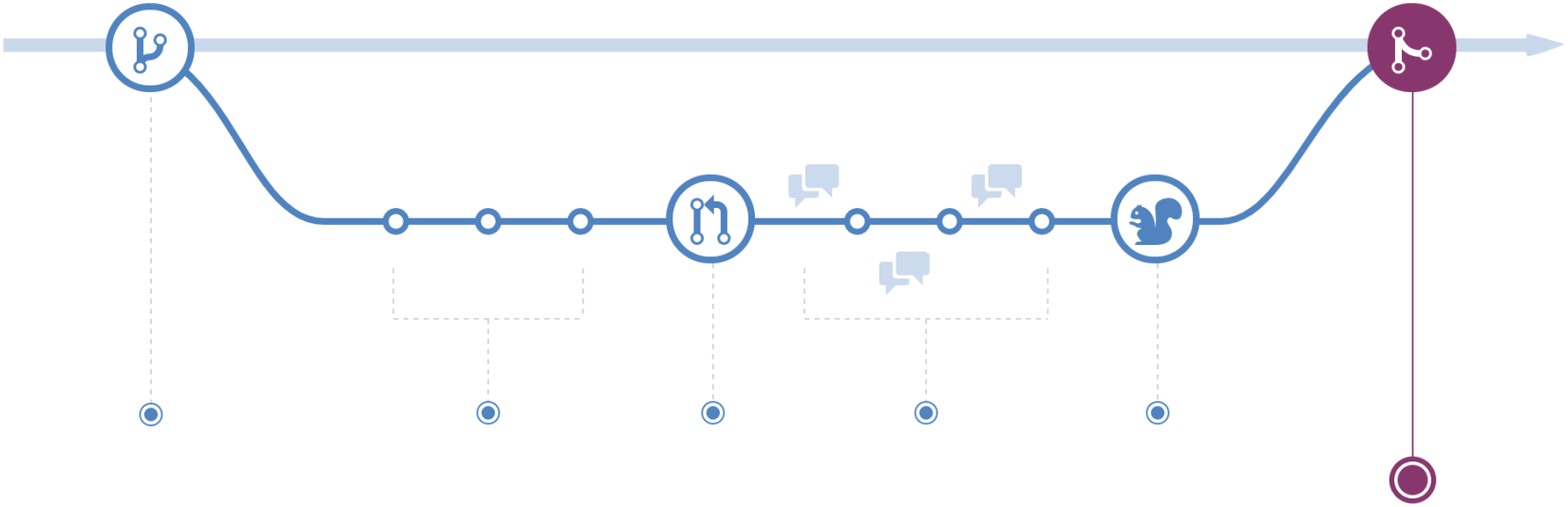
Continuous Release and Deployment



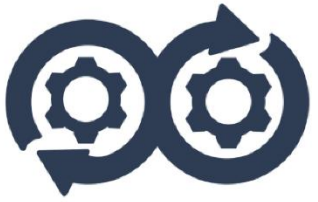
GitHub flow



1. Create a branch
2. Checkout branch and make changes
3. Open pull request
4. Peer review
5. Deploy
6. Merge into Master branch



Configuration Management



Configuration Drift

POTHOLE

- Your infrastructure requirements change
- Configuration of a server falls out of policy

- Manage with Infrastructure as Code (IAC)
  - Terraform
  - Ansible
  - Chef
  - Puppet

Manual

POP QUIZ: Manual tasks

What manual tasks do you deal with repeatedly?

Can they be automated?

Common manual tasks

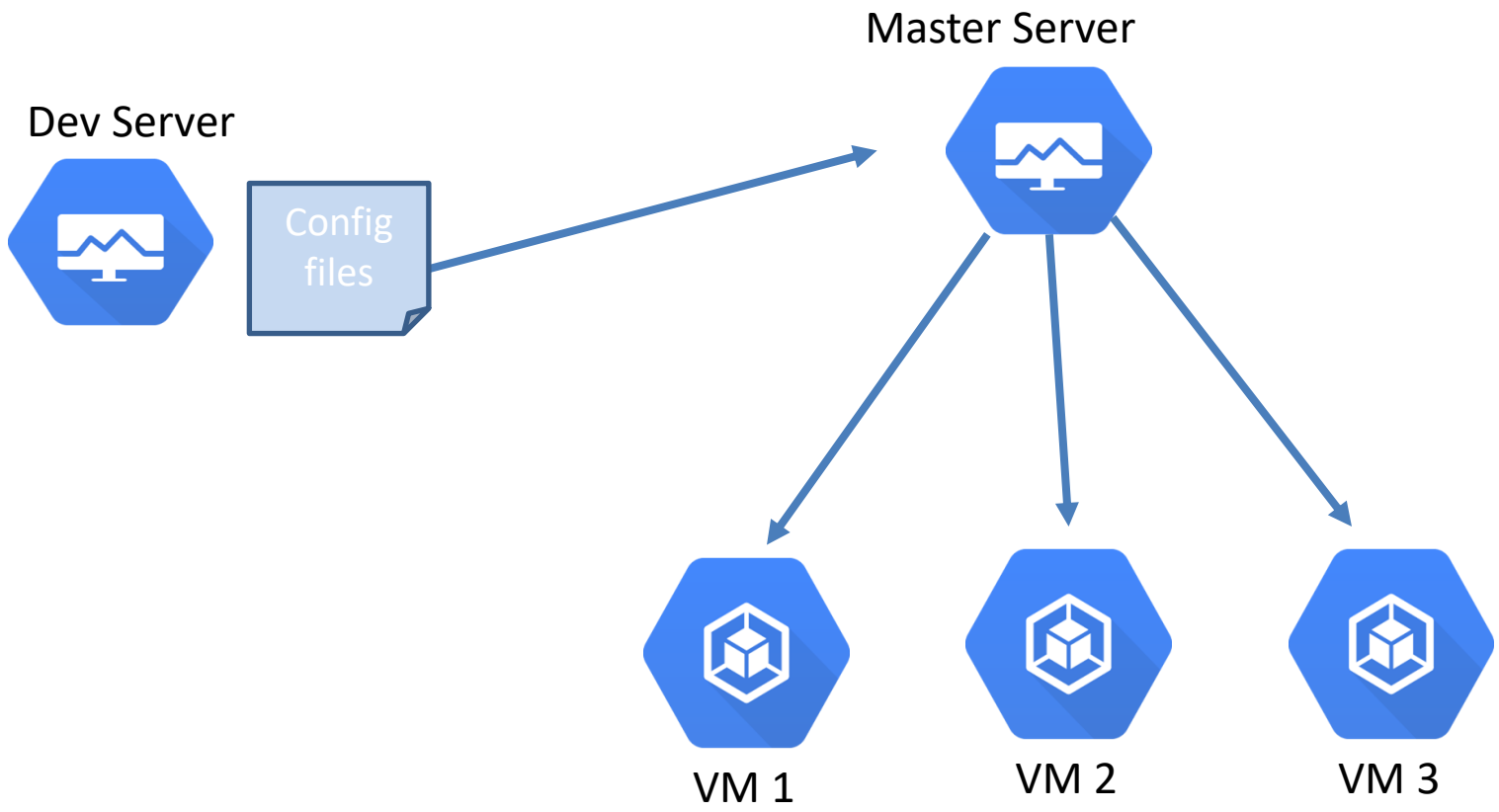
Installation and configuration:

- Operating System
- Runtime
- Libraries
- Utilities
- Configuration files

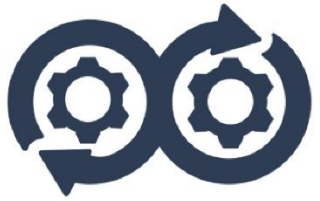
- Build application
- Test application
- Deploy application

Automated

Automated

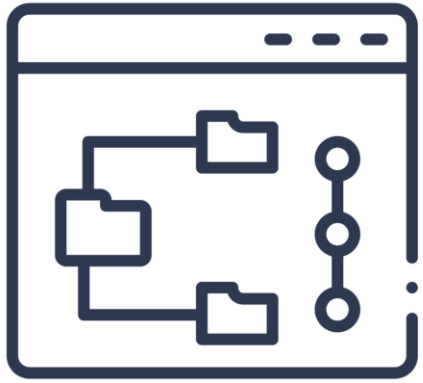


Terraform



Automating Infrastructure

Provisioning resources

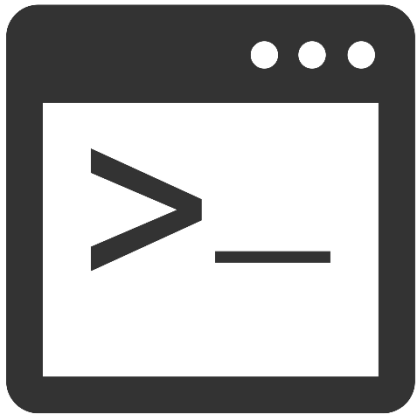


Version Control

Plan Updates

Reusable Templates

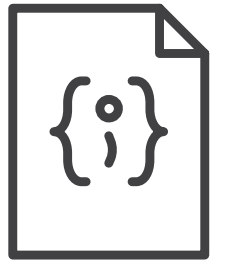
Terraform components



Terraform executable

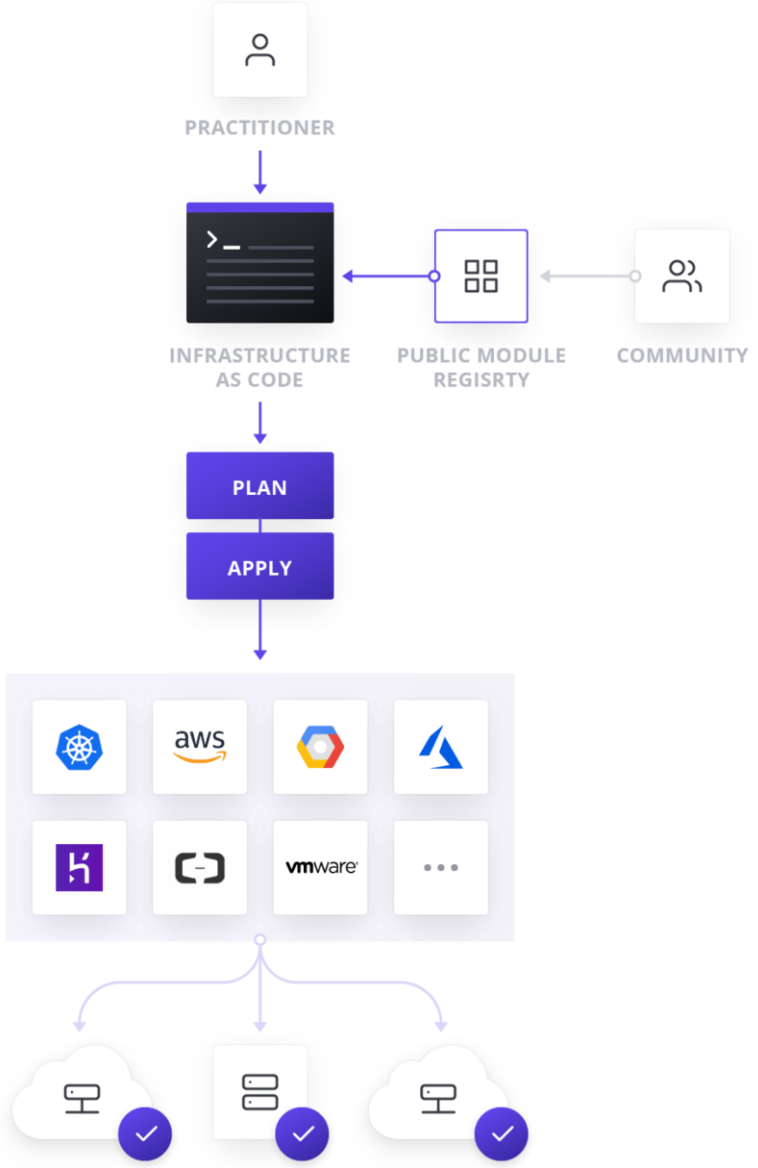
Terraform files

Terraform plugins

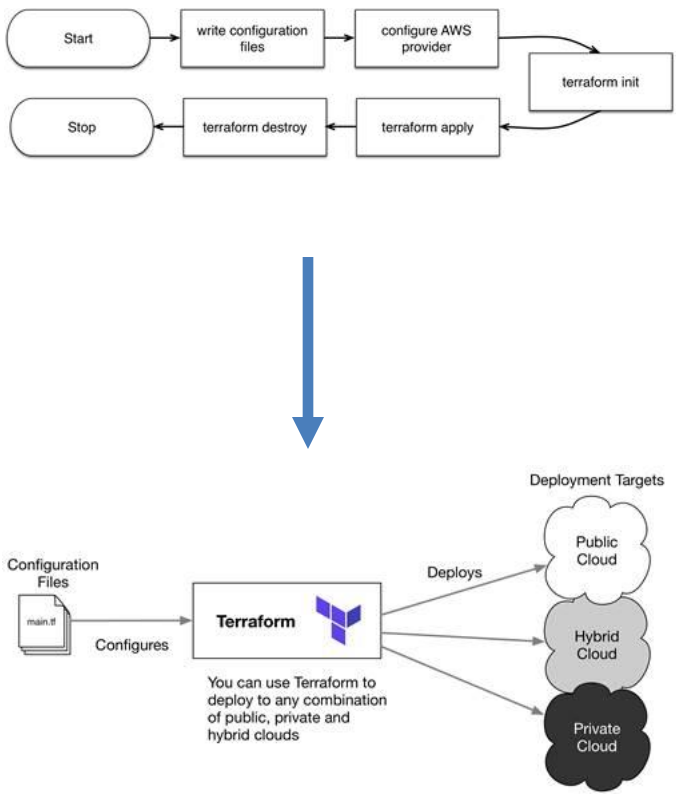


Terraform state

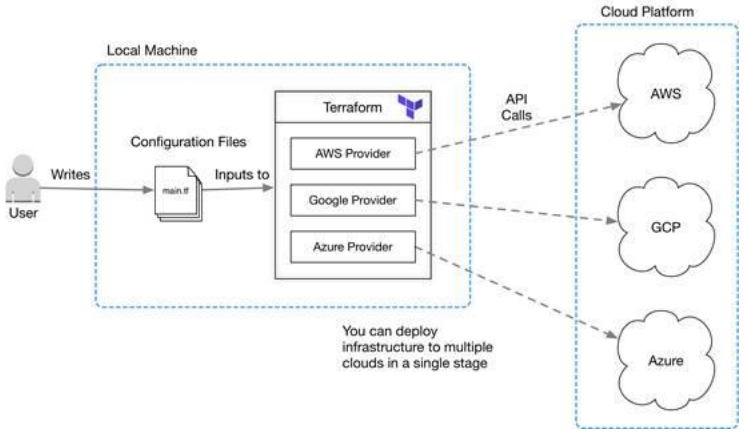
Terraform architecture

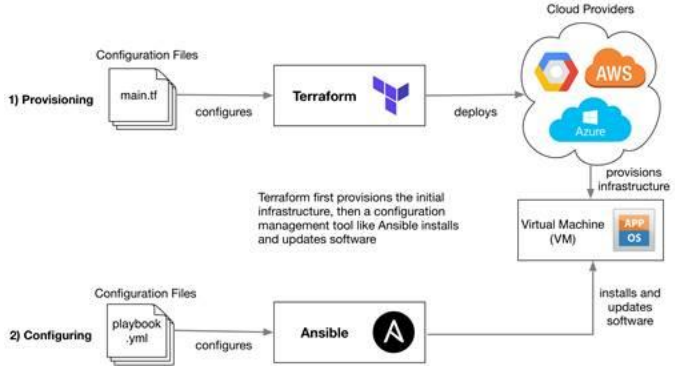


Terraform architecture

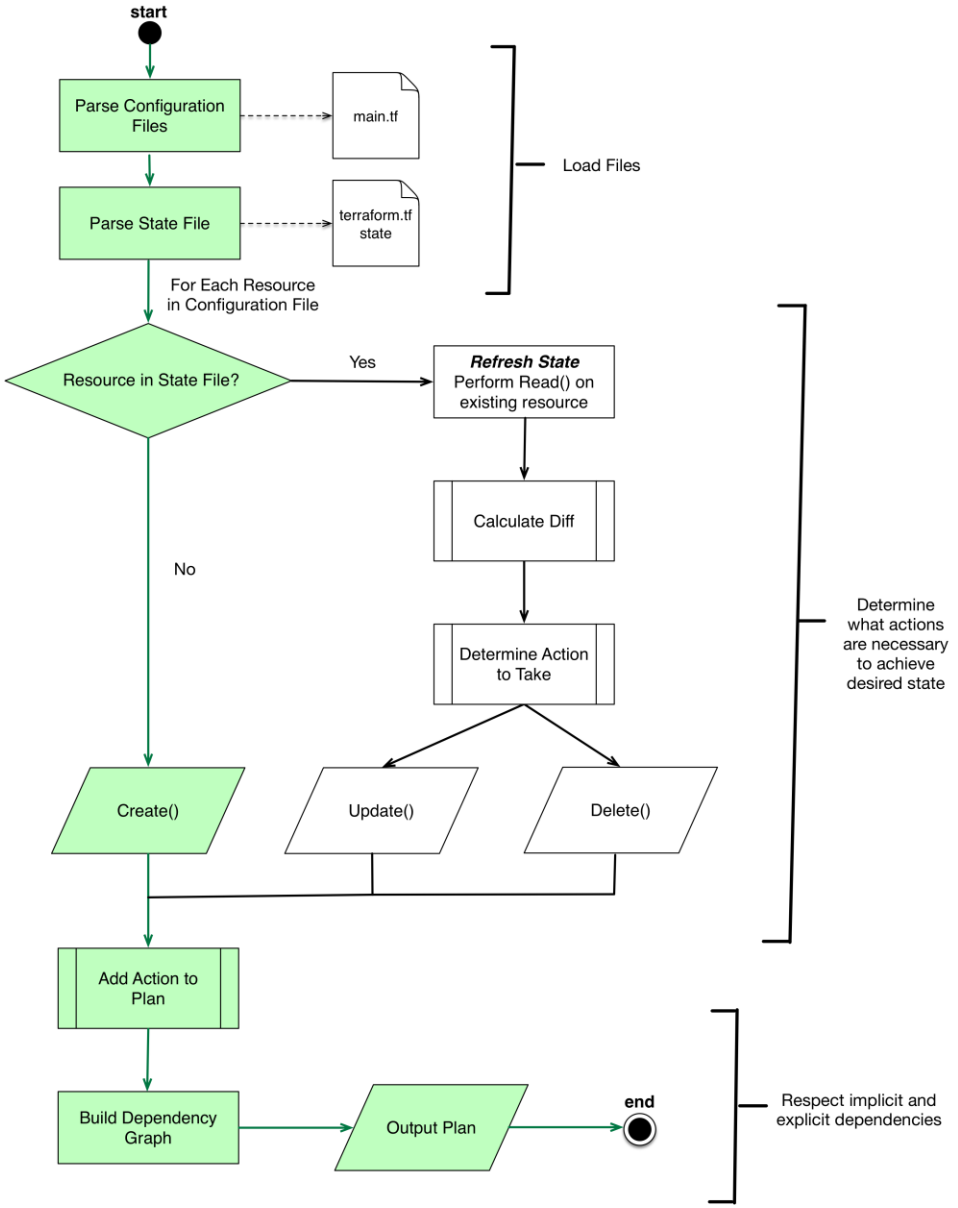


Terraform architecture





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 AWS API for 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"
}
```

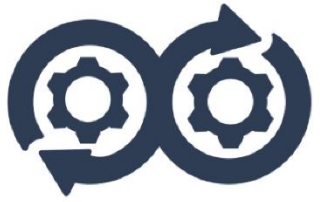
- Define sane defaults in variables.tf

Terraform files

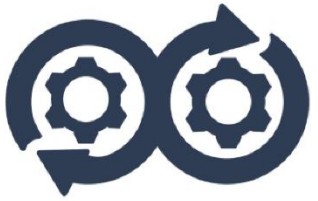
```
resource "aws_instance" "aws-k8s-master" {
  subnet_id              = "${var.aws_subnet_id}"
  depends_on             = ["aws_security_group.k8s_sg"]
  ami                    = "${data.aws_ami.ubuntu.id}"
  instance_type          = "${var.aws_instance_size}"
  vpc_security_group_ids = ["${aws_security_group.k8s_sg.id}"]
  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

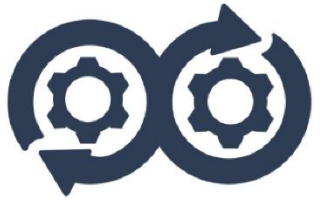
Demo: Terraform



Lab: Terraform Introduction

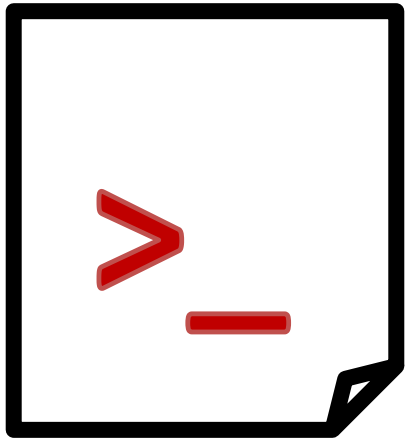


Ansible



Scenario Progression

1. Manual Commands

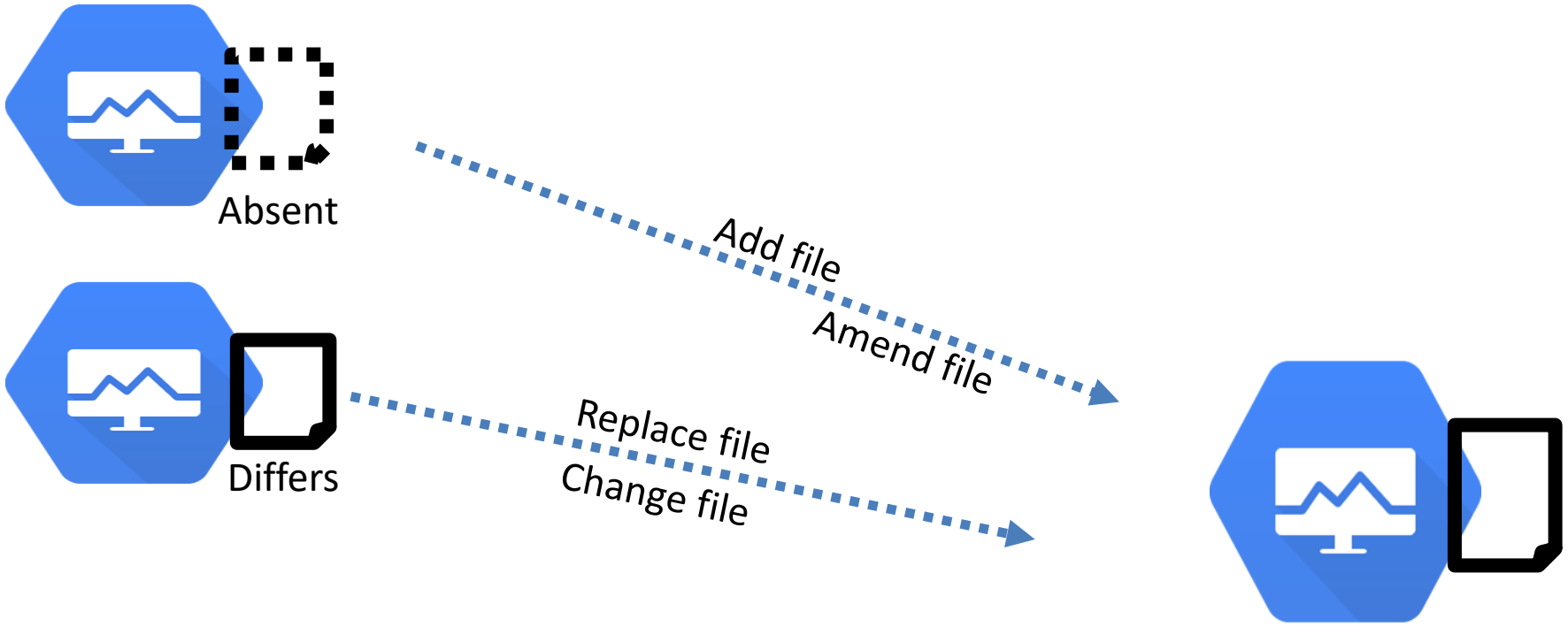


2. Reusable Scripts

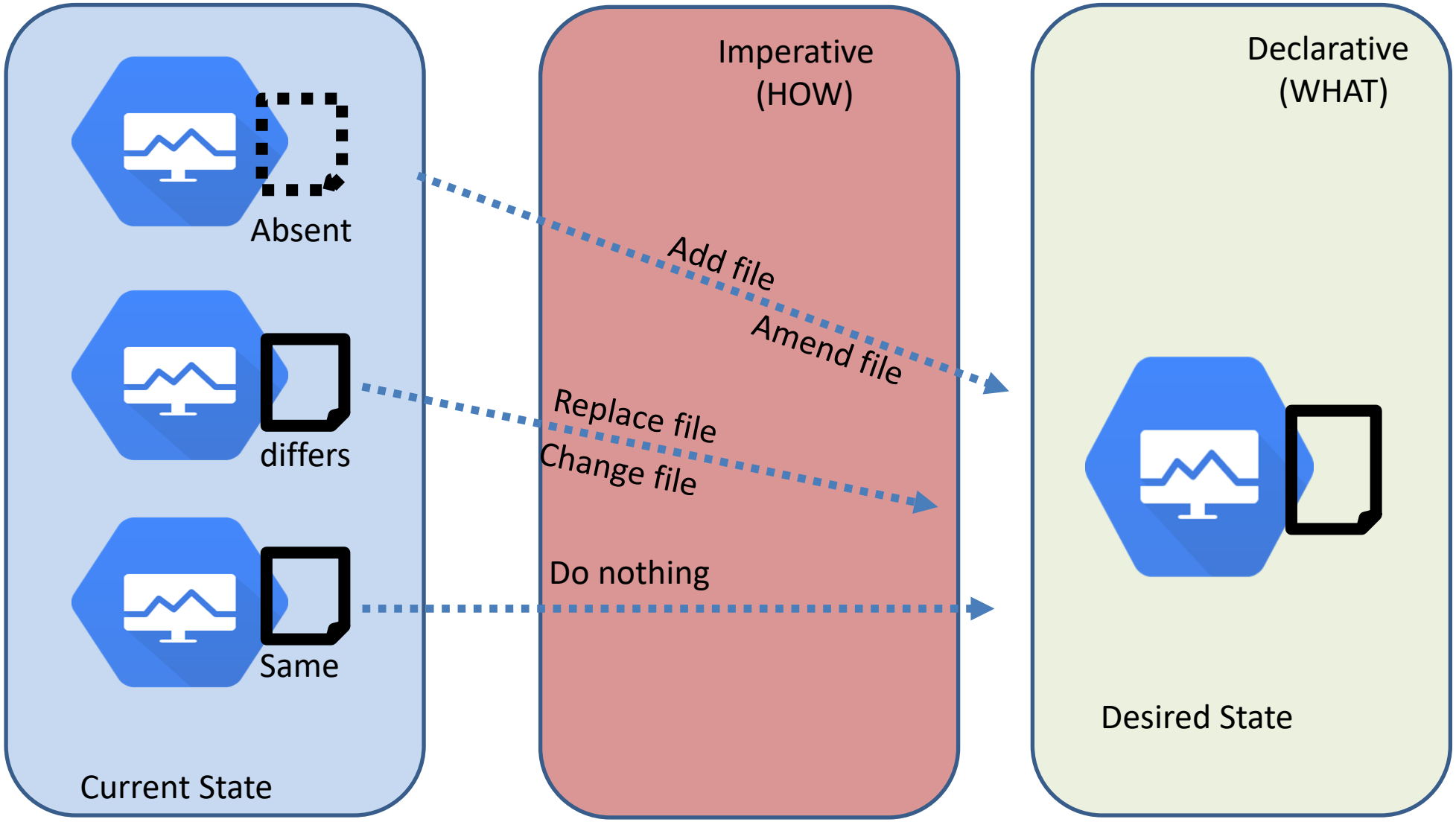
3. Ansible Ad-hoc

4. Playbooks

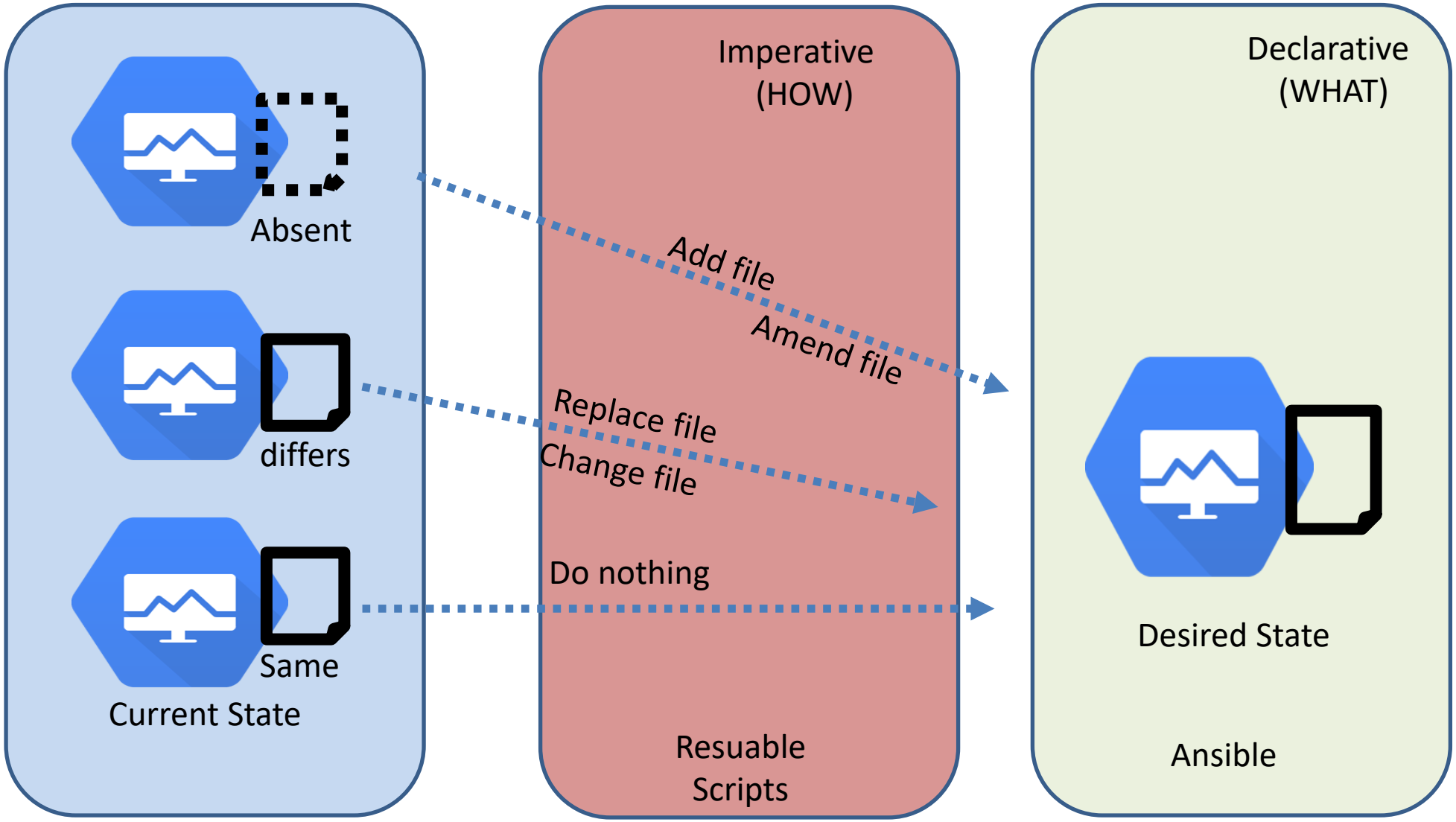
Reconciliation



Reconciliation



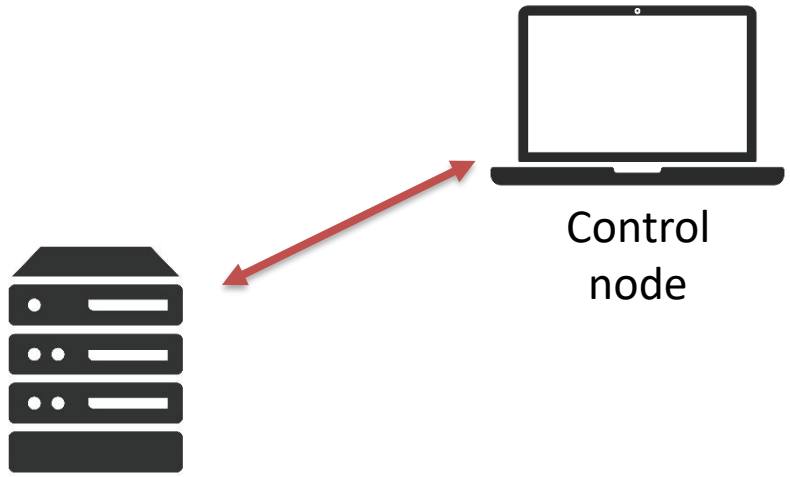
Reconciliation



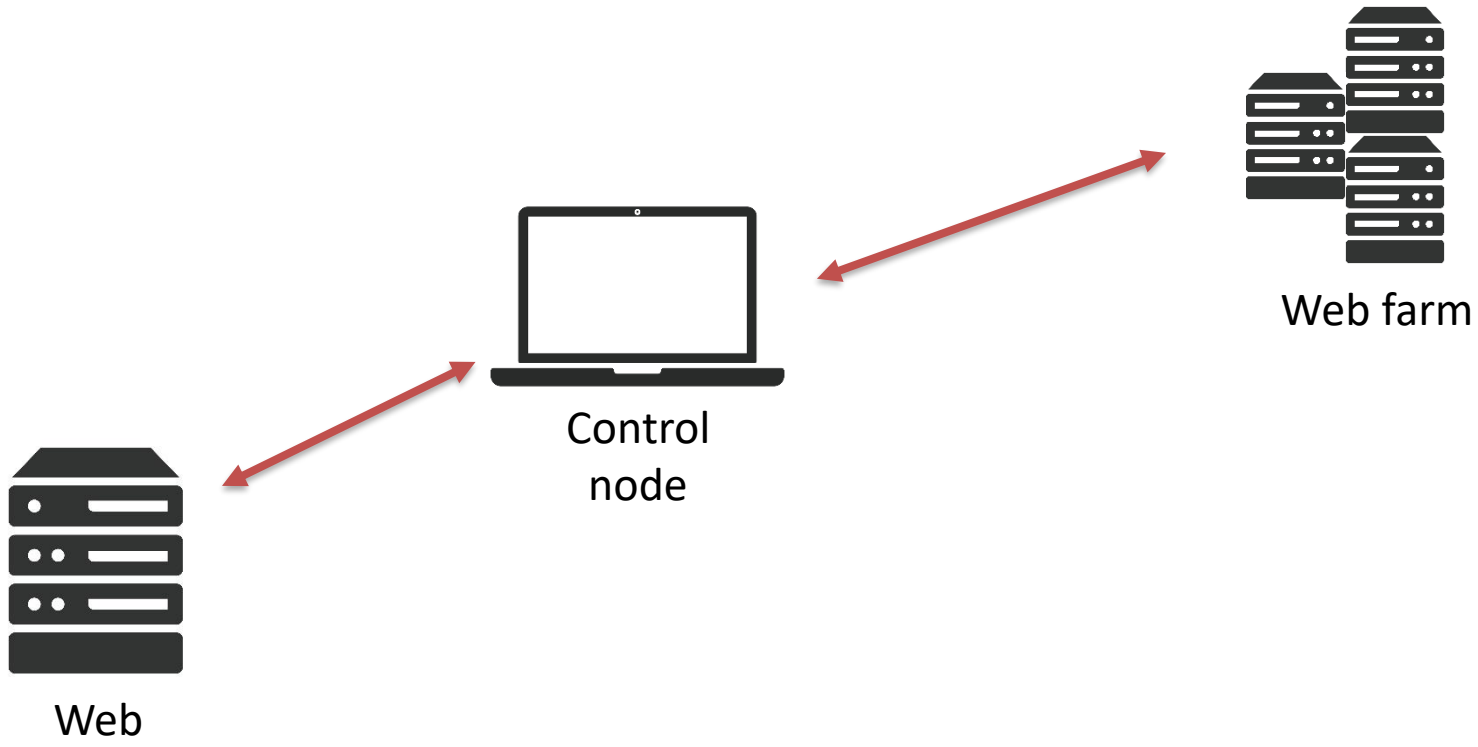
Architecture

Control
node

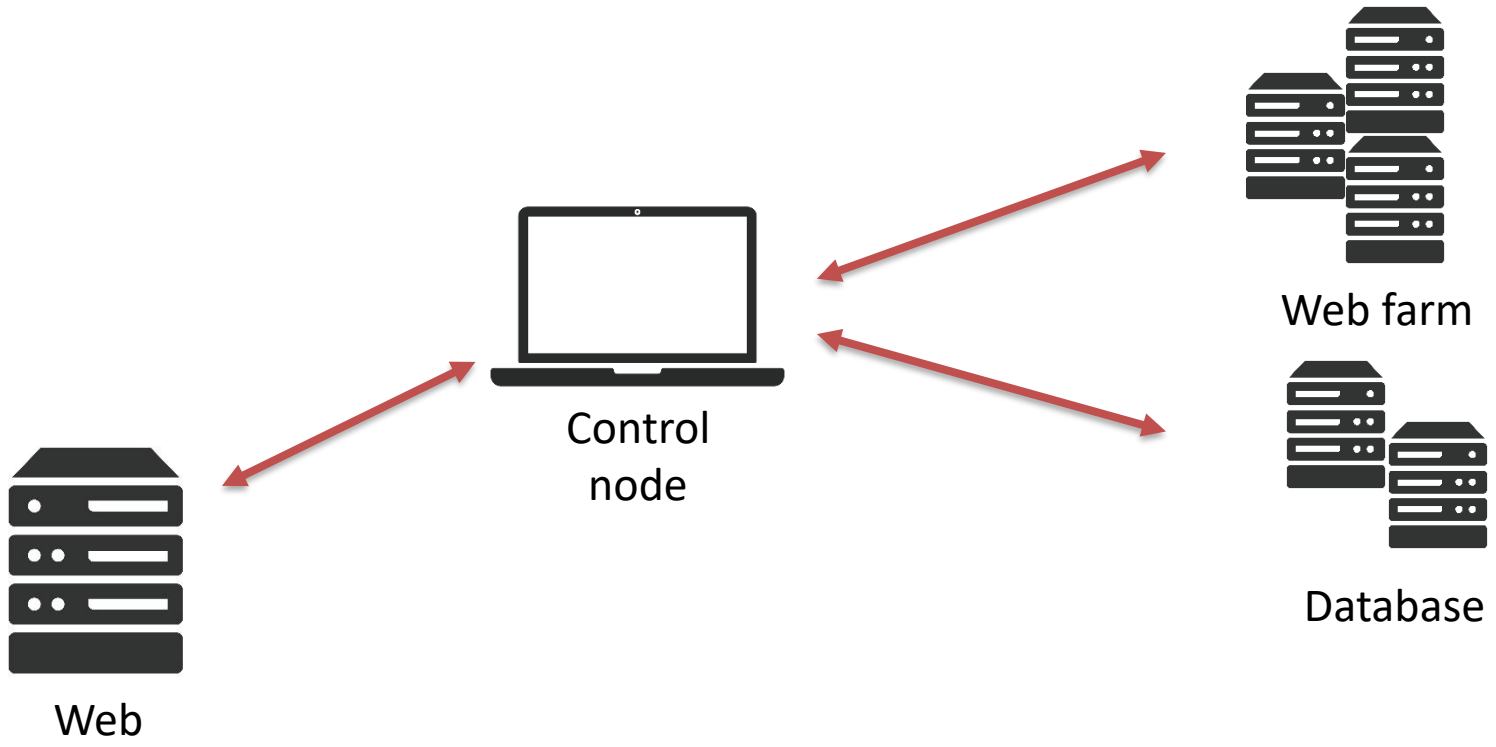
Architecture



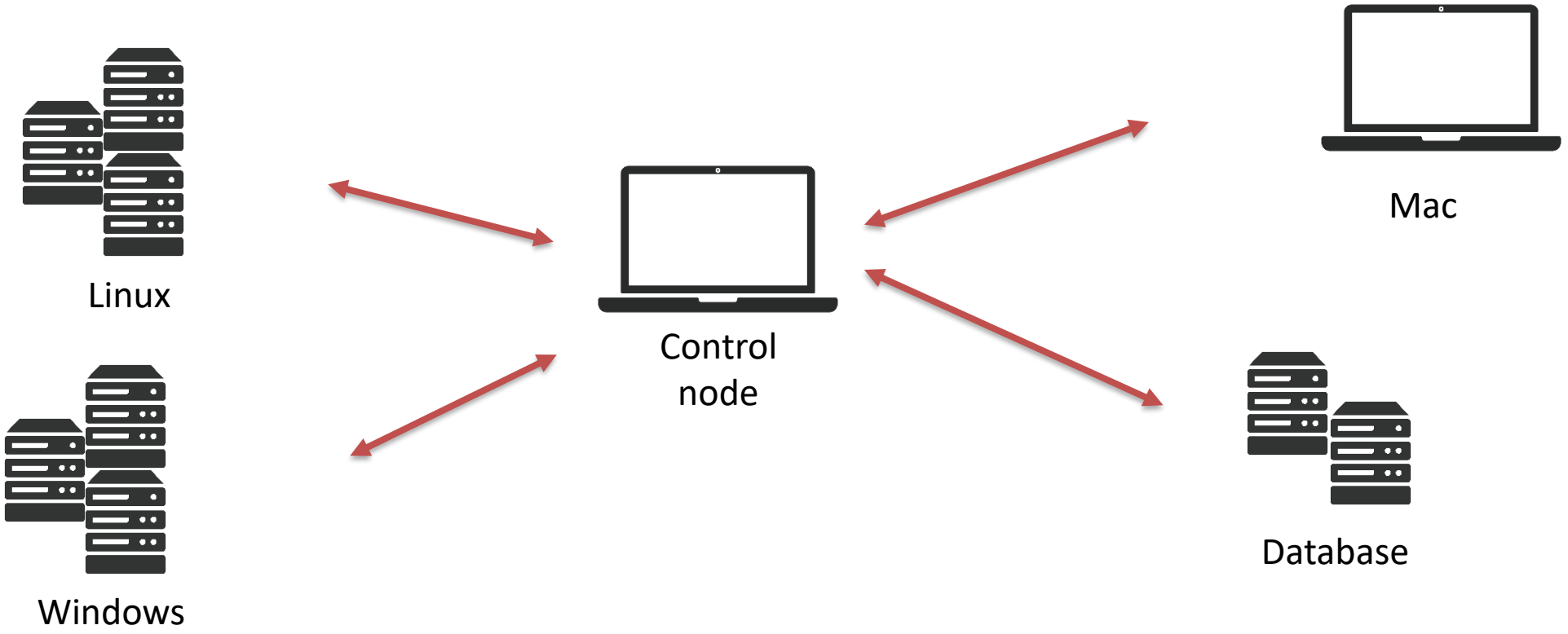
Architecture



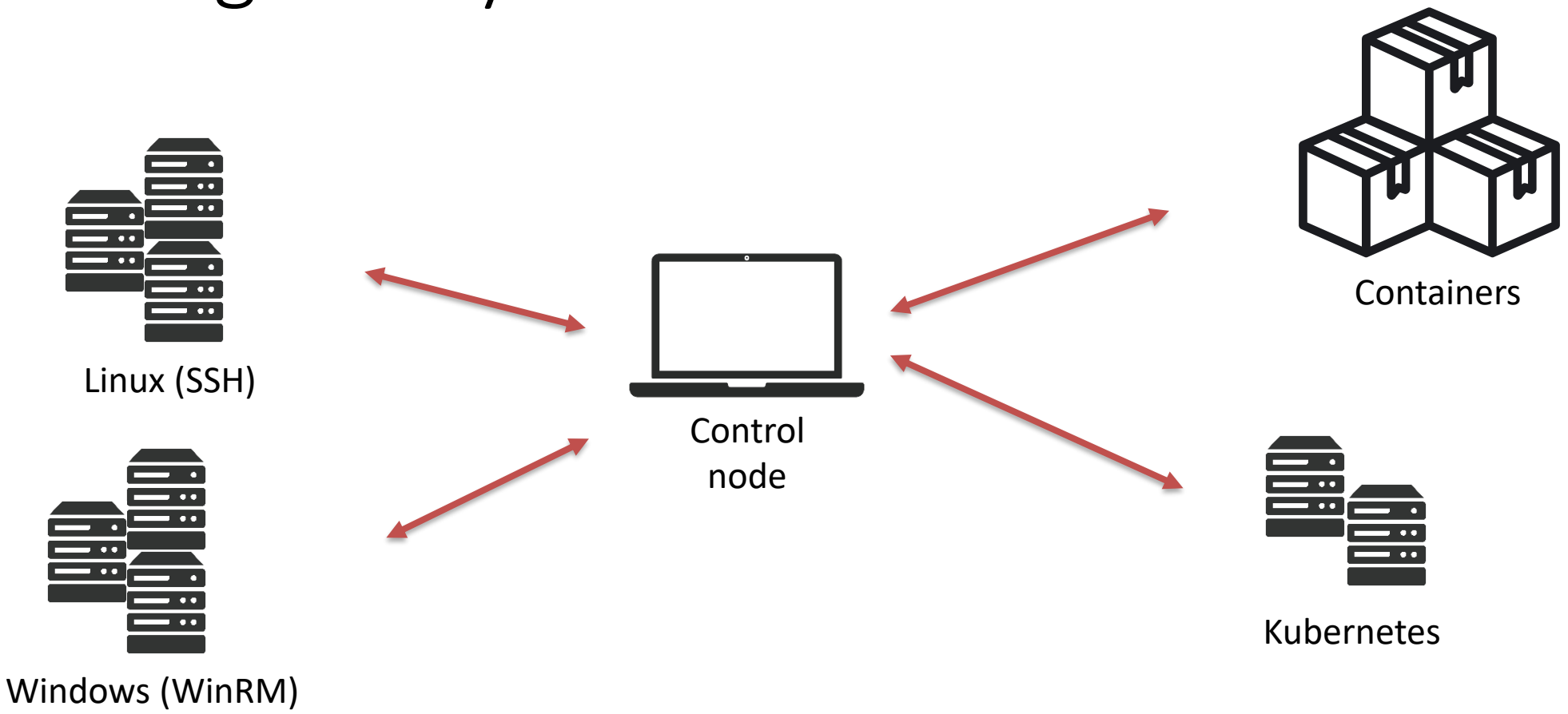
Architecture



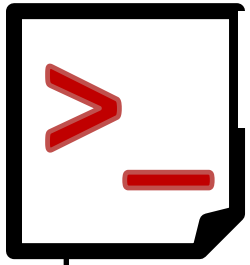
Connecting to anywhere



Connecting to anywhere



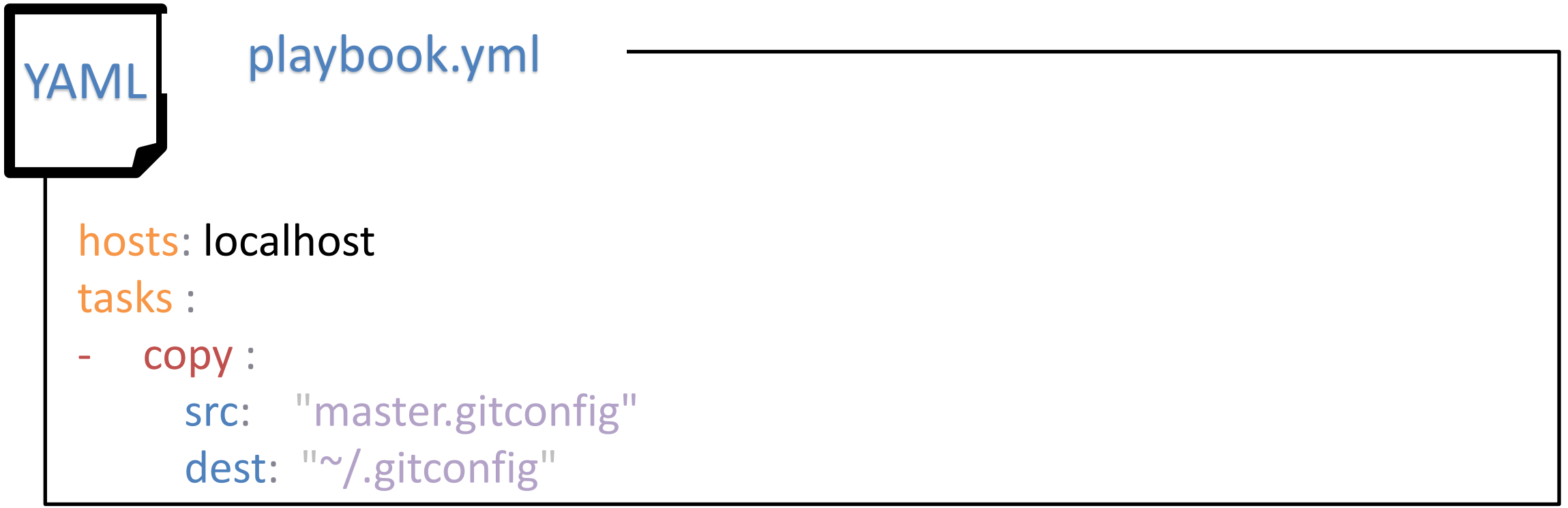
Ansible Ad-hoc



Scripted Ad-hoc

```
ansible -m copy -a "src=master.gitconfig dest=~/.gitconfig" localhost

ansible -m homebrew -a "name=bat state=latest" localhost

ansible -i hosts all -m ping -u ubuntu
```

Playbook



Playbook

playbook.yml

```
hosts: localhost
tasks :
-    copy :
        src:    "master.gitconfig"
        dest:  "~/.gitconfig"
```

>_

```
ansible-playbook playbook.yml
```

More complex playbook

```
- name: Install Docker
  hosts: tag_role_k8s_master:tag_role_k8s_member
  remote_user: ubuntu
  gather_facts: no
  handlers:
    - include: roles/handlers/main.yml
  tasks:
    - package:
        name: "docker-ce"
        state: latest
    - user:
        name: "{{ ansible_ssh_user }}"
        groups: docker
        append: yes
      notify: restart docker
```

Name of Play

More complex playbook

```yaml
- name: Install Docker
  hosts: tag_role_k8s_master:tag_role_k8s_member
  remote_user: ubuntu
  gather_facts: no
  handlers:
    - include: roles/handlers/main.yml
  tasks:
    - package:
        name: "docker-ce"
        state: latest
    - user:
        name: "{{ ansible_ssh_user }}"
        groups: docker
        append: yes
      notify: restart docker
```

Hosts to run Play on

More complex playbook

```
- name: Install Docker
  hosts: tag_role_k8s_master:tag_role_k8s_member
  remote_user: ubuntu
  gather_facts: no
  handlers:
    - include: roles/handlers/main.yml
  tasks:
    - package:
        name: "docker-ce"
        state: latest
    - user:
        name: "{{ ansible_ssh_user }}"
        groups: docker
        append: yes
      notify: restart docker
```

User to run Play as

More complex playbook

```
- name: Install Docker
  hosts: tag_role_k8s_master:tag_role_k8s_member
  remote_user: ubuntu
  gather_facts: no
  handlers:
    - include: roles/handlers/main.yml
  tasks:
    - package:
        name: "docker-ce"
        state: latest
    - user:
        name: "{{ ansible_ssh_user }}"
        groups: docker
        append: yes
      notify: restart docker
```

Define what to do after package is installed.

More complex playbook

```
- name: Install Docker
  hosts: tag_role_k8s_master:tag_role_k8s_member
  remote_user: ubuntu
  gather_facts: no
  handlers:
    - include: roles/handlers/main.yml
  tasks:
    - package:
        name: "docker-ce"
        state: latest
    - user:
        name: "{{ ansible_ssh_user }}"
        groups: docker
        append: yes
      notify: restart docker
```

Call the defined handler

More complex playbook

```yaml
- name: Install Docker
  hosts: tag_role_k8s_master:tag_role_k8s_member
  remote_user: ubuntu
  gather_facts: no
  handlers:
    - include: roles/handlers/main.yml
  tasks:
    - package:
        name: "docker-ce"
        state: latest
    - user:
        name: "{{ ansible_ssh_user }}"
        groups: docker
        append: yes
      notify: restart docker
```

Tasks to run
- install Docker

More complex playbook

```
- name: Install Docker
  hosts: tag_role_k8s_master:tag_role_k8s_member
  remote_user: ubuntu
  gather_facts: no
  handlers:
    - include: roles/handlers/main.yml
  tasks:
    - package:
        name: "docker-ce"
        state: latest
    - user:
        name: "{{ ansible_ssh_user }}"
        groups: docker
        append: yes
      notify: restart docker
```

Create user account

Lab: Ansible Introduction