ALGONQUIN COLLEGE

Introduction to Terraform

CST8918 - DevOps - IaC

C. Ayala - 23S

Introduction to Terraform

Unleashing the Potential of Infrastructure as Code: Empowering DevOps with Terraform.

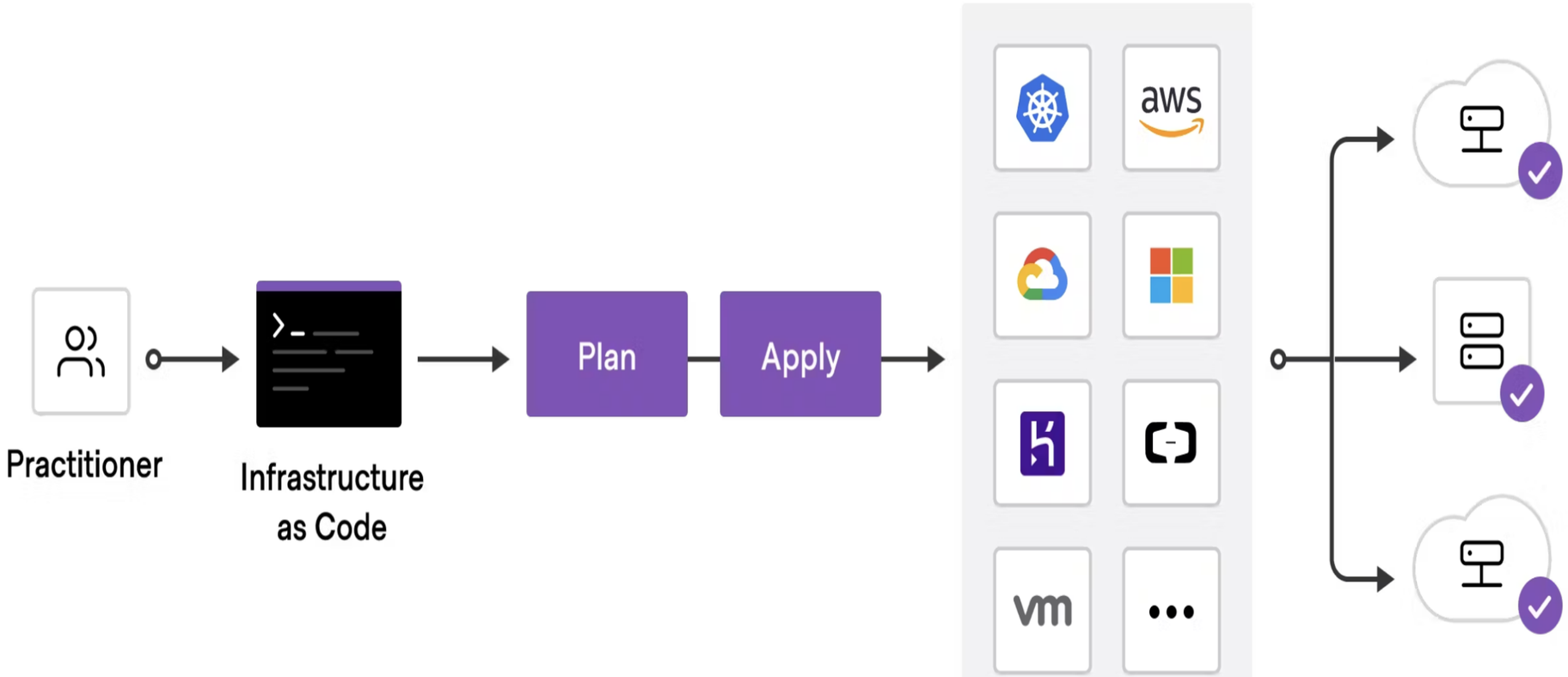
Introduction to Terraform

- Terraform: Open-source IaC software by HashiCorp
- Usage: Provisioning and management of infrastructure
- Benefits: Version control, reusability, multi-provider support
- Case Use: Managing multi-cloud deployments in an organization
- Pro Tip: Plan before applying changes to infrastructure
- Common Issue: State conflicts in team environments

Terraform's role in DevOps and IaC

- Enables infrastructure provisioning via config files
- Part of DevOps for automated deployments
- Central to IaC: Codifying infrastructure setup
- Promotes reproducibility and version control
- Fosters collaboration among development and ops

Terraform's Role in DevOps and IaC



Infographic of DevOps & IaC with Terraform highlighted

Source: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructur

Creation of Terraform Scripts

- Terraform scripts: provider, resources, variables
- Provider: Interfaces between Terraform and platform
- Resources: Components being managed
- Variables: Parameterize configuration, increase reusability
- Declarative nature: Defines desired states
- Scripts outline infrastructure provisioning instructions

Resource Configuration

- Resources represent infrastructure components
- Examples: virtual machines, databases, networks
- Configuration must match desired state

Variables and Output

- Variables allow parameterization, customization
- Variables are defined in scripts
- Output captures, displays infrastructure info

Writing Scripts for Different Stages

- Terraform scripts for different stages
- Adapt configs based on environment
- Flexibility, reusability of Terraform scripts

Managing Terraform Scripts

- Structure code with modules
- Keep resources & providers minimal
- Use version control systems
- Regularly run 'terraform fmt'
- Employ automated testing methods

Terraform Modules

- Modules: reusable, self-contained packages
- Promotes code reuse
- Reduces redundancy
- Example:

```
    Module "vpc" {
    Source = "../modules/vpc"
    Region = "ca-central"
}
```

Remote State Management and Dependencies

- Remote state: shares state data
  - Useful in team environments
- Resource dependencies:
  - Ensures proper order
  - Uses 'depends_on' keyword
- State management example:
  - Terraform backend 'S3'
  - Remote state data storage

Validating Scripts

- 'terraform validate': checks syntax
  - Ensures correct configuration
- Script validation:
  - Pre-execution check
  - Identifies potential errors
- Validation importance:
  - Prevents deployment issues
  - Improves code quality

'terraform validate' and 'terraform plan'

- 'terraform validate' command:
  - Pre-execution check
  - Example: 'terraform validate'
- 'terraform plan' command:
  - Generates execution plan
  - Example: 'terraform plan'

Importance of Validation

- Prevents errors
- Ensures consistency
- Makes deployment safe

Why Terraform

- Declarative syntax
- Provider diversity
- Module functionality
- State management
- One downside: learning curve

Automation Scripts for Scaling

- Scaling:
  - Adjusting capacity to meet demand
- Need for automation:
  - Efficiency
  - Accuracy
  - Quick response to changes
- Terraform for scaling:
  - Infrastructure as Code
  - Declarative syntax
  - Provider diversity

Recap of Major Points

- Terraform script structure: Provider, resources, variables
- Resource config: Represents infra components
- Terraform modules: Promotes code reusability
- Remote state management: Handles resource dependencies
- Terraform validation: Checks for syntax errors

Conclusion and Q&A

Ending the class and opening the floor for questions and answers.