Hands-on Infrastructure Automation with Terraform on AWS

Alexander Savchuk

Building a multi-tier environment

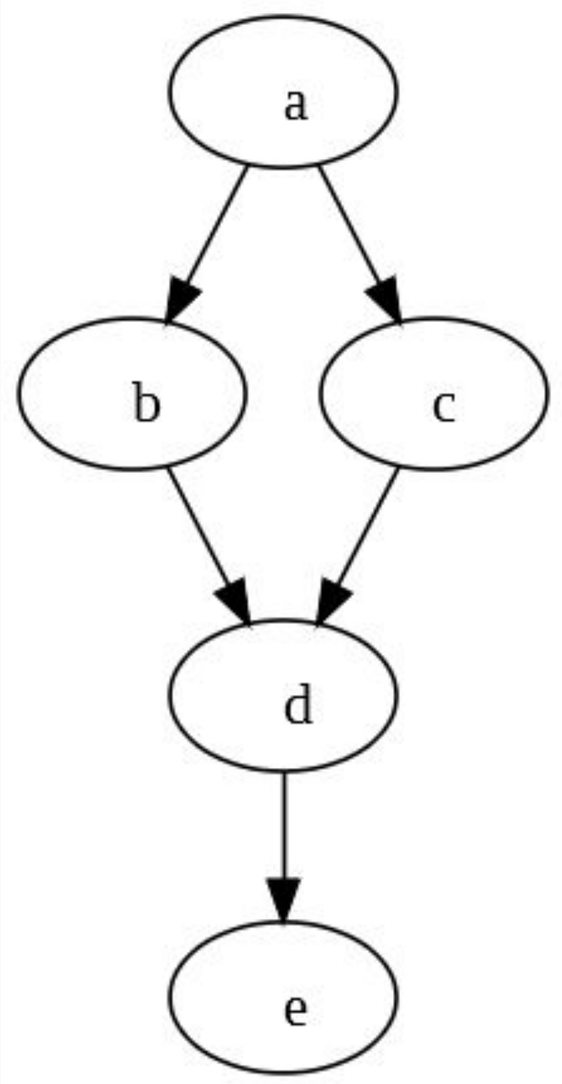
Section 4

Working with the dependency graph

Dependency graph

- All resources in the configuration are organised into a graph
- The graph is used to determine the order in which the resources are created

Directed acyclic graph



Parallelism

- Up to 10 nodes can be processed concurrently by default
- Configurable with **`-parallelism`** flag for `plan`, `apply`, and `destroy` commands (advanced setting)

Dependencies

- **Implicit**: one resource references another resource using the interpolation syntax

```
resource "aws_lb" "todo_app" {
  security_groups =      ["${aws_security_group.lb.id}"]
}
```

- **Explicit**: using **depends_on** metaparameter

```
depends_on = ["aws_security_group.lb"]
```

Use explicit dependencies to resolve race conditions

Main takeaways

- Use **outputs** and **remote state data source** to integrate stacks of resources in a complex environment
- Keep your code DRY by using `count` parameter and splat expressions
- Use templates to generate complex string inputs, such as user data scripts or ECS task definitions

Main takeaways

- Use outputs and remote state data source to integrate stacks of resources in a complex environment
- Keep your code DRY by using **`count` parameter** and **splat expressions**
- Use templates to generate complex string inputs, such as user data scripts or ECS task definitions

Main takeaways

- Use outputs and remote state data source to integrate stacks of resources in a complex environment
- Keep your code DRY by using `count` parameter and splat expressions
- Use **templates** to generate complex string inputs, such as user data scripts or ECS task definitions

Creating reusable components with modules

Next Section

Learning objective

Use Terraform templates to compose complex string inputs.