Explain Terraform vs. Other Software

- ✓ Terraform provides a flexible abstraction of resources and providers.
- ✓ This model allows for representing everything from physical hardware, virtual machines, and containers, to email and DNS providers.
- ✓ Because of this flexibility, Terraform can be used to solve many different problems.
- ✓ This means there are a number of existing tools that overlap with the capabilities of Terraform.
- ✓ We compare Terraform to a number of these tools, but it should be noted that Terraform is not mutually exclusive with other systems.
- ✓ It can be used to manage a single application, or the entire datacenter.

Terraform vs. Chef, Puppet, etc.

- ✓ Configuration management tools install and manage software on a machine that already exists.
- ✓ Terraform is not a configuration management tool, and it allows existing tooling to focus on their strengths: bootstrapping and initializing resources.
- ✓ Using provisioners, Terraform enables any configuration management tool to be used to setup a resource once it has been created.
- ✓ Terraform focuses on the higher-level abstraction of the datacenter and associated services, without sacrificing the ability to use configuration management tools to do what they do best.
- ✓ It also embraces the same codification that is responsible for the success of those tools, making entire infrastructure deployments easy and reliable.

Terraform vs. CloudFormation, Heat, etc.

- ✓ Tools like CloudFormation, Heat, etc. allow the details of an infrastructure to be codified into a configuration file.
- ✓ The configuration files allow the infrastructure to be elastically created, modified and destroyed.
- ✓ Terraform is inspired by the problems they solve.

- ✓ Terraform similarly uses configuration files to detail the infrastructure setup, but it goes further by being both cloud-agnostic and enabling multiple providers and services to be combined and composed.
- ✓ For example, Terraform can be used to orchestrate an AWS and OpenStack cluster simultaneously, while enabling 3rd-party providers like Cloudflare and DNSimple to be integrated to provide CDN and DNS services.
- ✓ This enables Terraform to represent and manage the entire infrastructure with its supporting services, instead of only the subset that exists within a single provider.
- ✓ It provides a single unified syntax, instead of requiring operators to use independent and non-interoperable tools for each platform and service.
- ✓ Terraform also separates the planning phase from the execution phase, by using the concept of an execution plan.
- ✓ By running terraform plan, the current state is refreshed and the configuration is consulted to generate an action plan.
- ✓ The plan includes all actions to be taken: which resources will be created, destroyed or modified.
- ✓ It can be inspected by operators to ensure it is exactly what is expected.
- ✓ Using terraform graph, the plan can be visualized to show dependent ordering.
- ✓ Once the plan is captured, the execution phase can be limited to only the actions in the plan.
- ✓ Other tools combine the planning and execution phases, meaning operators are forced to mentally reason about the effects of a change, which quickly becomes intractable in large infrastructures.
- ✓ Terraform lets operators apply changes with confidence, as they know exactly what will happen beforehand.

Terraform vs. Boto, Fog, etc.

- ✓ Libraries like Boto, Fog, etc. are used to provide native access to cloud providers and services by using their APIs.
- ✓ Some libraries are focused on specific clouds, while others attempt to bridge them all and mask the semantic differences.

- ✓ Using a client library only provides low-level access to APIs, requiring application developers to create their own tooling to build and manage their infrastructure.
- ✓ Terraform is not intended to give low-level programmatic access to providers, but instead provides a high level syntax for describing how cloud resources and services should be created, provisioned, and combined.
- ✓ Terraform is very flexible, using a plugin-based model to support providers and provisioners, giving it the ability to support almost any service that exposes APIs.

Terraform vs. Custom Solutions

- ✓ Most organizations start by manually managing infrastructure through simple scripts or web-based interfaces.
- ✓ As the infrastructure grows, any manual approach to management becomes both error-prone and tedious, and many organizations begin to home-roll tooling to help automate the mechanical processes involved.
- ✓ These tools require time and resources to build and maintain.
- ✓ As tools of necessity, they represent the minimum viable features needed by an organization, being built to handle only the immediate needs.
- ✓ As a result, they are often hard to extend and difficult to maintain.
- ✓ Because the tooling must be updated in lockstep with any new features or infrastructure, it becomes the limiting factor for how quickly the infrastructure can evolve.
- ✓ Terraform is designed to tackle these challenges. It provides a simple, unified syntax, allowing almost any resource to be managed without learning new tooling.
- ✓ By capturing all the resources required, the dependencies between them can be resolved automatically so that operators do not need to remember and reason about them.
- ✓ Removing the burden of building the tool allows operators to focus on their infrastructure and not the tooling.
- ✓ Furthermore, Terraform is an open source tool.
- ✓ In addition to HashiCorp, the community around Terraform helps to extend its features, fix bugs and document new use cases.

- ✓ Terraform helps solve a problem that exists in every organization and provides a standard that can be adopted to avoid reinventing the wheel between and within organizations.
- ✓ Its open source nature ensures it will be around in the long term.