What are the use cases of Terraform?

Use Cases:

1. Heroku App Setup

- ✓ Heroku is a popular PaaS for hosting web apps.
- ✓ Developers create an app, and then attach add-ons, such as a database, or email provider.
- ✓ One of the best features is the ability to elastically scale the number of dynos or workers.
- ✓ However, most non-trivial applications quickly need many add-ons and external services.
- ✓ Terraform can be used to codify the setup required for a Heroku application, ensuring that all the required add-ons are available,
- ✓ but it can go even further: configuring DNSimple to set a CNAME, or setting up Cloudflare as a CDN for the app.
- ✓ Best of all, Terraform can do all of this in under 30 seconds without using a web interface.

2. Multi-Tier Applications

- ✓ A very common pattern is the N-tier architecture.
- ✓ The most common 2-tier architecture is a pool of web servers that use a database tier.
- ✓ Additional tiers get added for API servers, caching servers, routing meshes, etc.
- ✓ This pattern is used because the tiers can be scaled independently and provide a separation of concerns.
- ✓ Terraform is an ideal tool for building and managing these infrastructures.
- ✓ Each tier can be described as a collection of resources, and the dependencies between each tier are handled automatically;
- ✓ Terraform will ensure the database tier is available before the web servers are started and that the load balancers are aware of the web nodes.
- ✓ Each tier can then be scaled easily using Terraform by modifying a single count configuration value.

- ✓ Because the creation and provisioning of a resource is codified and automated, elastically scaling with load becomes trivial.

3. Self-Service Clusters

- ✓ At a certain organizational size, it becomes very challenging for a centralized operations team to manage a large and growing infrastructure.
- ✓ Instead it becomes more attractive to make "self-serve" infrastructure, allowing product teams to manage their own infrastructure using tooling provided by the central operations team.
- ✓ Using Terraform, the knowledge of how to build and scale a service can be codified in a configuration.
- ✓ Terraform configurations can be shared within an organization enabling customer teams to use the configuration as a black box and use Terraform as a tool to manage their services.

4. Software Demos

- ✓ Modern software is increasingly networked and distributed.
- ✓ Although tools like Vagrant exist to build virtualized environments for demos, it is still very challenging to demo software on real infrastructure which more closely matches production environments.
- ✓ Software writers can provide a Terraform configuration to create, provision and bootstrap a demo on cloud providers like AWS.
- ✓ This allows end users to easily demo the software on their own infrastructure, and even enables tweaking parameters like cluster size to more rigorously test tools at any scale.

5. Disposable Environments

- ✓ It is common practice to have both a production and staging or QA environment.
- ✓ These environments are smaller clones of their production counterpart, but are used to test new applications before releasing in production.
- ✓ As the production environment grows larger and more complex, it becomes increasingly onerous to maintain an up-to-date staging environment.

- ✓ Using Terraform, the production environment can be codified and then shared with staging, QA or dev.
- ✓ These configurations can be used to rapidly spin up new environments to test in, and then be easily disposed of.
- ✓ Terraform can help tame the difficulty of maintaining parallel environments, and makes it practical to elastically create and destroy them.

6. Software Defined Networking

- ✓ Software Defined Networking (SDN) is becoming increasingly prevalent in the datacenter, as it provides more control to operators and developers and allows the network to better support the applications running on top.
- ✓ Most SDN implementations have a control layer and infrastructure layer.
- ✓ Terraform can be used to codify the configuration for software defined networks.
- ✓ This configuration can then be used by Terraform to automatically setup and modify settings by interfacing with the control layer.
- ✓ This allows configuration to be versioned and changes to be automated.
- ✓ As an example, AWS VPC is one of the most commonly used SDN implementations, and can be configured by Terraform.

7. Resource Schedulers

- ✓ In large-scale infrastructures, static assignment of applications to machines becomes increasingly challenging.
- ✓ To solve that problem, there are a number of schedulers like Borg, Mesos, YARN, and Kubernetes.
- ✓ These can be used to dynamically schedule Docker containers, Hadoop, Spark, and many other software tools.
- ✓ Terraform is not limited to physical providers like AWS.
- ✓ Resource schedulers can be treated as a provider, enabling Terraform to request resources from them.
- ✓ This allows Terraform to be used in layers: to setup the physical infrastructure running the schedulers as well as provisioning onto the scheduled grid.

8. Multi-Cloud Deployment

- ✓ It's often attractive to spread infrastructure across multiple clouds to increase fault-tolerance.
- ✓ By using only a single region or cloud provider, fault tolerance is limited by the availability of that provider.
- ✓ Having a multi-cloud deployment allows for more graceful recovery of the loss of a region or entire provider.
- ✓ Realizing multi-cloud deployments can be very challenging as many existing tools for infrastructure management are cloud-specific.
- ✓ Terraform is cloud-agnostic and allows a single configuration to be used to manage multiple providers, and to even handle cross-cloud dependencies.
- ✓ This simplifies management and orchestration, helping operators build large-scale multi-cloud infrastructures.