HashiCorp Terraform

Terraform Enterprise Operations

TFE Path to Production

Weeks 1 & 2

Terraform Instance Deployed

- TFE overview & Architectural deep dive

Weeks 3 & 4

Terraform Workflow

- Importing Resources, Migrating State & Terraform Workflows

Week 5

Office Hours

- Open Forum for Q&A

Week 6

Terraform Governance

- Terraform Governance & Integrations

Week 7

Ready for Production!

- Operating your Terraform Instance
- Exit ramp and operational readiness

Agenda

01

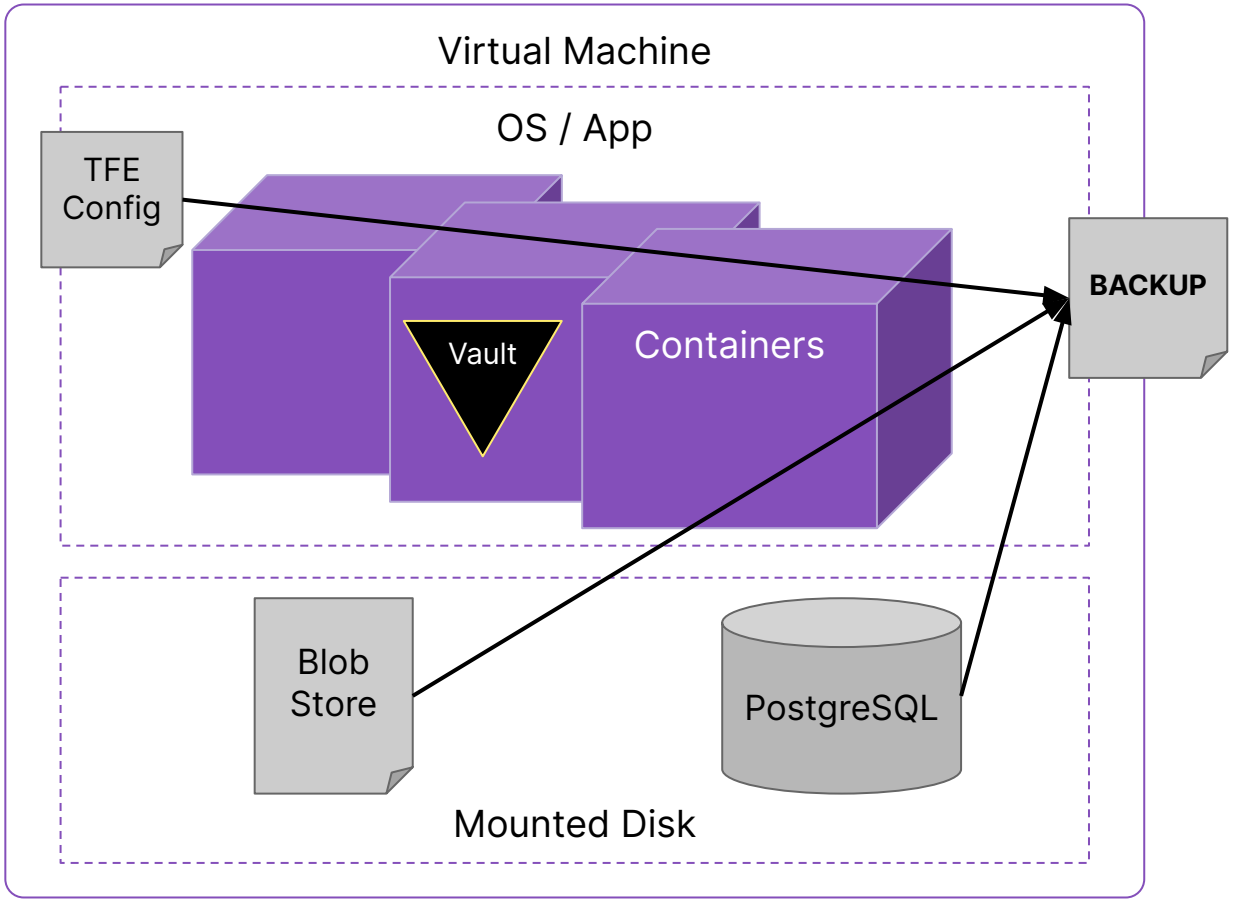
Backup & Restore

TFE Backup & Restoration

- A Terraform Enterprise (TFE) deployment is made up of a number of services including: a Postgres database, object store, & Vault cluster
- Consider the database & object storage as one conceptual data layer for TFE even though they are technically separate
- TFE includes built in tooling to assist with backing up all the services
- TFE's operational mode dictates how each component of TFE is backed up

Mounted Disk

- PostgreSQL Database and Blob Storage use mounted disks for their data
- Backup and restore of those volumes is the responsibility of the user
- *Vault Data* is stored in PostgreSQL and accordingly lives on the mounted disk
- If the instance running Terraform Enterprise is lost, the use of mounted disks means no state data is lost

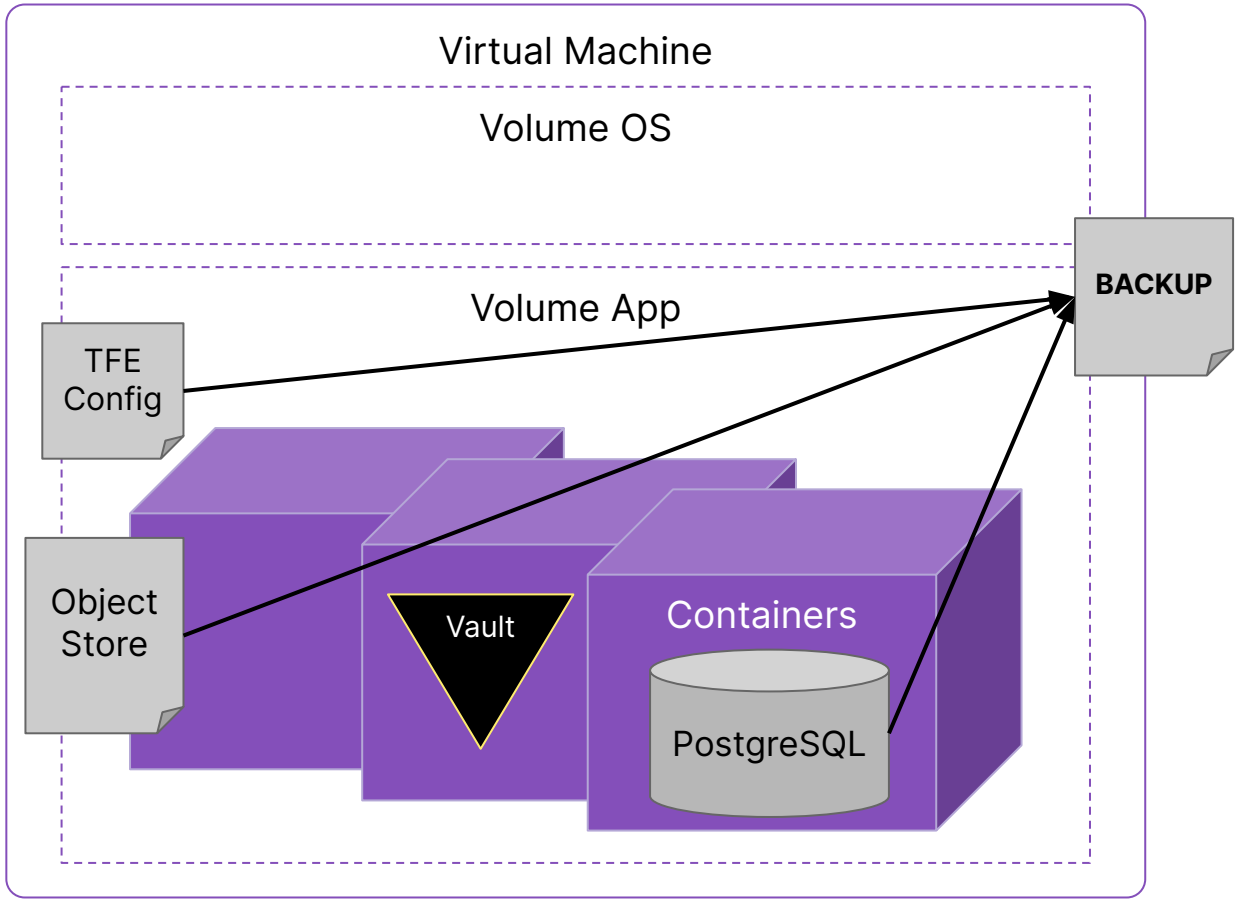


Mounted Disk Backup/Restore

- Backing up a TFE instance running Mounted Disk operational mode simply entails correctly backing up the Virtual Machine that houses the instance
- Backups need to encapsulate the entire VM contents at a single point in time to ensure the integrity of the machine and its attached data disk
- Each cloud provider has native backup/snapshot tool, using them is the recommended pattern (AWS, Azure, GCP)
- VMware patterns depend upon data disk configuration
  a. **Isolated data disk:** take regular snapshots/backups of the data disk, replicate disk to redundant DCs
  b. **Shared data disk:** use SAN native snapshot and replication tools, enable versioning if the SAN volumes supports it

External Services

- The Application Layer and Coordination Layer execute on a Linux instance
- Storage Layer is external PostgreSQL server & S3-compatible Storage
- Maintenance & backup of PostgreSQL and S3-compatible storage are managed by the operator(s)



External Services Backup /Restore

- TFE backup API can be utilized for smaller implementations
- Use of cloud-native tooling for day-to-day backup and recovery
- When Preparing to backup refer to your cloud provider for recommendations for the following:
  - Application Server - Recommended to automatically replace nodes when a node or availability zone fails
  - Object Store - Choose fast storage optimized for use, scales well, and automatically replicates to another zone in the same region
  - Database - For high availability in a single public cloud region, recommend deploying the database in a multi-availability zone configuration to add resilience against *recoverable* outages

Backup and Restore Best Practices

- Harden the server image using CIS benchmarking
- Run TFE on single-use machines
- Deploy immutable instances
- Pin the version of TFE that the Replicated install.sh script deploys to avoid accidental version upgrades
- Document the backup and restoration process
- Arrange for staff who did *not* write the documentation process to run a test restore using it
- Regularly test the backup and restoration process to ensure the documentation is reliable

Backup & Restore

Built-in Methods

1. **Automated Snapshots**

   Recommended automated snapshots

2. **Backup API**

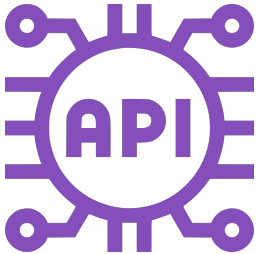
   Terraform Enterprise API to backup and restore all application data

Backup & Restore

Built-in Methods

Automated Snapshots

Recommended automated snapshots



Backup API

Terraform Enterprise API to backup and restore all application data

Automated Snapshots

- There can be two types of data on the snapshots of the TFE instance
  - Terraform Enterprise application data: Core product data such as run history, configuration history, & state history which changes frequently
  - Terraform Enterprise installer data: Data used to configure Terraform Enterprise itself such as installation type, database connection settings, & hostname which rarely changes
- In the Mounted Disk and External Services operational modes, only installer data is stored on the instance
- Daily snapshots are recommended for Mounted Disk and External services
- Automated snapshots are most effective when using mounted disk or External Services as the amount of backed up data is smaller and less risky

Automated Recovery

- Replicated version 2.17.0 or greater is required to use the restore mechanism
  - The version can be checked using replicatedctl version command
- Provision a new Terraform Enterprise instance using the latest snapshot
- Examples of restore scripts
- Air Gap recovery considerations:
  - Version of Replicated is 2.31.0
  - license file and airgap package must be in place on the new instance prior to restore
  - The snapshot being used must also be from an airgap instance
  - airgap installation instructions

TFE Backup API

- The backup API backs up all of the data stored in a TFE installation, including both the blob storage and the PostgreSQL database
- The API is separate from the main application-level APIs and uses a different token
- Backup API is the only supported way to migrate between operational modes (mounted disk, external services)
- Best suited for smaller TFE installs
- Once backup is initiated, best practice will be to send the request from a server colocated with the Terraform Enterprise installation for best performance and to avoid disconnections

TFE API Restore

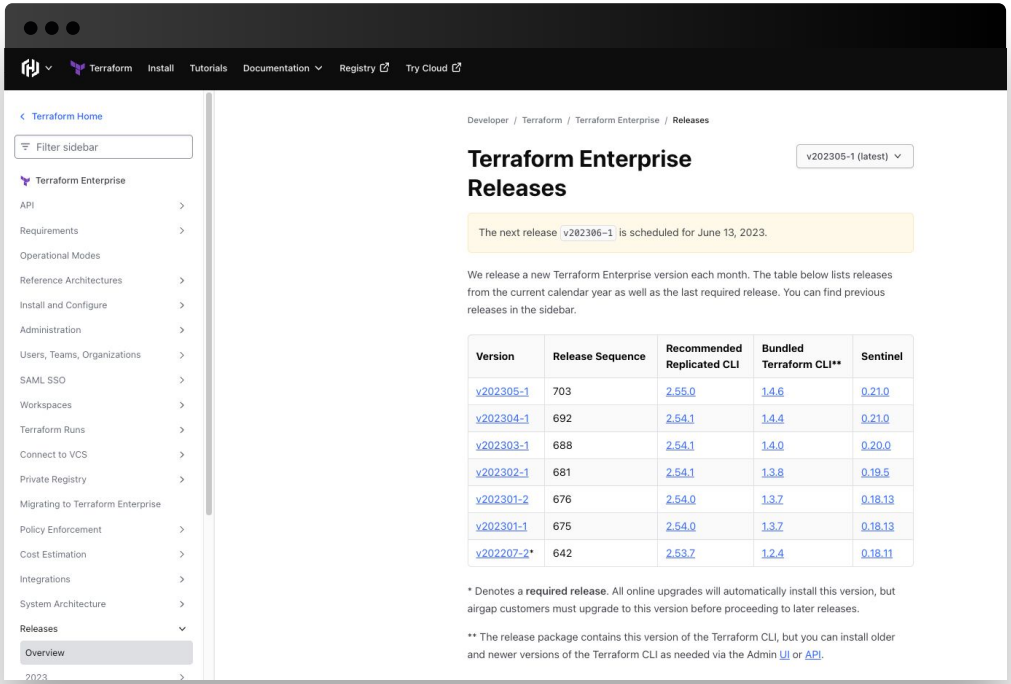
- A new TFE installation must be created before initiating a restore
- Due to large amount of data being uploaded, best practice will be to send the restore request from a server colocated with the TFE installation to avoid poor performance and disconnections
- Once restore is complete, the application can be restarted via the Install dashboard or CLI

02

Updates & Upgrades

TFE Updates / Upgrades

- TFE currently releases application updates on a monthly basis
- Updates include application features, bug fixes, & security updates
- Certain releases are ***required***, marked with an asterisk on the list of releases
- Required releases typically perform an internal data migration, and must not be bypassed



Upgrade Options

Immutable VMs

- Reduce risk by enabling offline testing
- Reduce chance of creep in OS image standards and TFE configuration, all stored as code in Source Control
- A bit more initial overhead and skill required (e.g. Packer and cloud-init)
- Automated, consistent process
- Operator is responsible for installing required releases

In Place

- OS patching and TFE updates are done in place
- Automatically installs required releases as part of the upgrade path
- High chance of drift from initial OS standards and TFE configuration over time
- Manual process with risk of human error
- Less skill required, easier for orgs with less mature IaC capabilities

Immutable VMs - Upgrade Types

OS Update / Upgrade

- TFE Should be installed after the approved OS image is created
- OS image just includes TFE dependencies and security hardening
- TFE install should be run using cloud-init
- Template can be tested using a copy of the TFE volume

TFE Update / Upgrade

- TFE update should use the same process as OS Update, just update the versions of Replicated and / or TFE run by cloud-init
- Testing is the same, take a copy of the TFE volume and run against that

Immutable VMs: Update/Upgrade Steps

1. Cloud Run in dev / test environment first!
2. Update the VM image to target desired versions of Replicated and TFE
3. Optionally, test offline using a volume snapshot
4. Schedule maintenance window
5. Stop access to TFE (e.g. take out of load balancer)
6. Take a backup of 3 components (config, blob, SQL)
7. Deploy the new image and let the update run against the live volume
8. Update the load balancer config to make the new instance available
9. Test
10. Repeat steps 4 to 9 in production environment

In Place: Update/Upgrade Steps

1. Run in ‘dev’ environment first!
2. Schedule maintenance window
3. Stop access to TFE (e.g. via load balancer)
4. Take a Backup of 3 components (config, blob, SQL)
5. Run the TFE Update Process
6. Run the Replicated Update Process
7. Restore access to TFE
8. Test
9. Repeat steps 2 to 8 in production

Update / Upgrade

Preparation

- Updates / Upgrades should be tested in a non-prod/dev environment prior to deploying in Production
- **Downtime is to be expected** during Upgrades / Updates
- Anticipated downtime windows range from 90 seconds to 30+ minutes depending on database updates involved
- Key upgrade prerequisites:
  - Read the Terraform Enterprise Release Notes
  - Export Terraform Enterprise Configuration
  - Determine which releases are required
  - Ensure you have a **viable, verified backup** prior to starting the process

Upgrade Process – UI

Navigate to **https://<TFE HOSTNAME>:8800/dashboard**

↓

Click “Check Now”

↓

Click “View Update”

↓

Click “Install Update”

↓

TFE will then automatically download and install the update. This usually takes 15-30 min depending on system and network performance

Upgrade Process - Airgapped - UI

Download airgap package to the update path location
Update path can be found here:
https://<TFE HOSTNAME>:8800/console/settings

↓

Navigate to **https://<TFE HOSTNAME>:8800/dashboard**

↓

Click “Check Now”

↓

Click “View Update”

↓

Click “Install Update”

↓

TFE will then automatically download and install the update. This usually takes 15-30 min depending on system and network performance

Upgrade Process - CLI

Connect to the Terraform Enterprise host machine using SSH

↓

Fetch the versions of Terraform Enterprise

```
replicatedctl app-release ls --fetch
```

↓

Upgrade to the latest version of Terraform Enterprise

```
replicatedctl app-release apply
```

Alternatively, upgrade to a specific version of Terraform Enterprise

```
replicatedctl app-release apply --sequence "504"
```

Upgrade Process Replicated Components

Connect to the Terraform Enterprise host machine using SSH

Re-run the Terraform Enterprise installation script for online installations

```
curl https://install.terraform.io/ptfe/stable | sudo bash
```

Airgapped:

Download the latest airgapped components and from a shell on your instance, in the directory where you placed the latest.tar.gz installer bootstrapper:

```
tar xzf latest.tar.gz
```

Re-run the Terraform Enterprise installation script for online installations

```
sudo ./install.sh airgap
```

Upgrade/Troubleshooting

- Terraform Enterprise typically has new major version releases on a monthly schedule with new feature sets and bug fixes included
- Between major releases, there are occasionally patch releases made available through the same release channel
- The upgrade procedures of Terraform Enterprise are available on this public document
- Additional resources:
  - Support Knowledge Base
  - Upgrading Terraform Enterprise
- Logging support tickets in the support portal

03

Telemetry & Monitoring

Monitoring Patterns

- Monitoring the health of TFE instances in Production Environments should include all three of the following patterns:
  1. Time-series Telemetry Data
  2. Log Forwarding & Analytics
  3. Health Checks
- Health check endpoints are useful for instance availability status & monitoring
- Performance metrics monitoring & alerting should be configured for the VM running TFE

Monitoring

- TFE provides an external health check endpoint on each instance
- Health check will return **`200 OK`** when TFE is up
- Health check endpoint is available at **`/_health_check`** with 2 modes:
  - Full check **`/_health_check?full=1`**
  - Minimal check **`/_health_check`**
- Health check endpoint will run a full check at instance startup & then minimal checks during normal operation
- Full health checks are heavier operations and increase system load and latency over minimal checks

Monitoring

Example External Health Check

```
$ curl http://$(docker inspect ptfe_health_check|jq -r
.
[].NetworkSettings.Networks[].IPAddress):23005/_health_
check

{"passed":true,"checks":[{"name":"Archivist Health
Check","passed":true},{"name":"Terraform Enterprise
Health Check","passed":true},{"name":"Terraform
Enterprise Vault Health
Check","passed":true},{"name":"Fluent Bit Health
Check","passed":false,"skipped":true},{"name":"RabbitMQ
Health Check","passed":true},{"name":"Vault Server
Health Check","passed":true}]}
```


Monitoring

Internal Health Check

- TFE provides an internal health check which is not dependant upon external network connectivity
- Internal health checks verify the following:
  - Archivist is up and healthy
  - The application can communicate with Redis & Postgres
  - The application can communicate with Vault & is able to encrypt and decrypt tokens
  - Test that RabbitMQ is able to send & consume messages
  - Verify that the Vault server is healthy

Monitoring

Example Internal Health Check

```
$ tfe-admin health-check

checking: Archivist Health Check...
|  checks that Archivist is up and healthy
|- ✓ PASS

checking: Terraform Enterprise Health Check...
|  checks that Terraform Enterprise is up and can communicate with Redis
and Postgres
|- ✓ PASS

checking: Terraform Enterprise Vault Health Check...
|  checks that Terraform Enterprise can connect to Vault and is able to
encrypt and decrypt tokens
|- ✓ PASS

checking: RabbitMQ Health Check...
|  checks that RabbitMQ can be connected to and that we can send and
consume messages
|- ✓ PASS

checking: Vault Server Health Check...
|  checks that the configured Vault Server is healthy
|- ✓ PASS

All checks passed.
```


Monitoring

Metrics & Telemetry

- Monitoring & alerting of standard server metrics should encapsulate:
  - I/O
  - CPU
  - RAM
  - Disk
- Telemetry from TFE is best stored in a metrics aggregation platform to collect durable metrics and collect trends
- Monitoring of Postgres instances should be configured when running in external services mode
- Monitoring of Postgres & Redis instances should be configured when running in Active/Active mode
- TFE supports Container Metrics (release v202201-1+)

Monitoring

Metrics

- TFE's metrics service aggregates data on a 5 second interval and retains data in memory for 15 seconds
- Metrics collection is disabled by default and must be enabled by setting the `metrics_endpoint_enabled` config flag to `"1"`
- JSON and Prometheus formats are supported
- Scrape interval should be set below 15s to ensure all data points are captured
- Ports for HTTP (9090) and HTTPS (9091) are configurable

Monitoring

Container Metrics

- TFE runtime container metrics report information about container instances
- Metadata labels added to each container metric
  - `id`
  - `name`
  - `image`
- Build worker metrics include additional labels
  - `run_type`
  - `run_id`
  - `workspace_name`
  - `organization_name`
- Global metrics
  - `tfe.run.count`
  - `tfe.run.limit`
  - `tfe.run.current.count`

Monitoring

Grafana Dashboard

Terraform Enterprise

A Grafana dashboard for Terraform Enterprise

Overview Revisions Reviews

Terraform Enterprise Monitoring Dashboard

Maintained by the Terraform Enterprise team at HashiCorp. Displays container resource utilization metrics for Terraform Enterprise components, including CPU usage, memory allocation, disk I/O, and network I/O, as well as run pipeline metrics.

For more information about the metrics exposed by Terraform Enterprise, see the documentation at https://www.terraform.io/enterprise/admin/infrastructure/monitoring#metrics-telemetry.

The queries used by the panels in this dashboard were created with a five second `scrape_interval` and `evaluation_interval` in Prometheus . The panels may not generate data successfully if your Prometheus configuration is set otherwise. This dashboard is provided as an example for how you could utilize the metrics that are exported out of TFE, but is not meant to be production ready.

Log Forwarding

- TFE supports forwarding logs to 1 or more external destinations
- Audit logs are emitted alongside application logs and contain the `[Audit Log]` string for differentiation
- Log forwarding is disabled by default, set `log_forwarding_enabled` to `“1”` enables forwarding
- Log forwarding requires:
  - Terraform Enterprise running on an instance using `systemd-journald`
  - A version of Docker that supports the `journald` logging driver
  - Network connectivity between TFE & external destination(s) where logs should be forwarded

Log Forwarding

Supported External Destinations

- Amazon CloudWatch
- Amazon S3
- Azure Blob Storage
- Azure Log Analytics
- Datadog
- Forward
- Google Cloud Platform Cloud Logging
- Splunk Enterprise HTTP Event Collector (HEC)
- Syslog

Audit Logging

```
2021-08-31 04:58:30 [INFO] [7a233ad1-c50c-4737-a925-3be901e55fcb] [Audit
Log]
{
    "resource":"run",
    "action":"create",
    "resource_id":"run-nL77p69bsesoF3RK",
    "organization":"example-org",
    "organization_id":"org-pveSPvxocni226Fn",
    "actor":"example-user",
    "timestamp":"2021-08-31T04:58:30Z",
    "actor_ip":"19.115.231.192"
}
```

Log Rotation

- Log forwarding uses the **`journald`** Docker logging driver & sends logs to **`systemd-journald`**
- Log forwarding can cause increased utilization for **`/var/log/journal`**
- To limit disk utilization of Log forwarding, configure **`SystemMaxFileSize & SystemMaxFiles`** settings in **`/etc/systemd/journald.conf`**

```
[Journal]
SystemMaxFileSize=1024M
SystemMaxFiles=7
```

Next Steps

Tutorials

https://developer.hashicorp.com/terraform/tutorials

Step-by-step guides to accelerate deployment of Terraform Enterprise

Terraform Install Tutorials Documentation Registry Try Cloud Search ⌘/ctrl K

Terraform Home

Tutorials

Get Started
AWS
Azure
Docker
GCP
OCI
Terraform Cloud

Fundamentals
CLI
Configuration Language
Modules
Provision
State
Terraform Cloud

Use Cases
Applications

Get Started

Build, change, and destroy infrastructure with Terraform. Start here to learn the basics of Terraform with your favorite cloud provider.

8 tutorials
aws
Build, change, and destroy AWS infrastructure using Terraform. Step-by-step, command-line tutorials will walk...

8 tutorials
Microsoft Azure
Build, change, and destroy Azure infrastructure using Terraform. Step-by-step, command-line tutorials will walk...

10 tutorials
Terraform Cloud
Collaborate on version-controlled configuration using Terraform Cloud. Follow this track to build, change, and...

7 tutorials
docker
Build, change, and destroy Docker infrastructure using Terraform. Step-by-step, command-line tutorials will walk...

7 tutorials
Google Cloud
Build, change, and destroy Google Cloud Platform (GCP) infrastructure using Terraform. Step-by-step, command-lin...

7 tutorials
ORACLE
Build, change, and destroy a virtual cloud network and subnet on Oracle Cloud Infrastructure (OCI) using Terraform....

New Tutorials

Try the newest tutorials for common Terraform tasks and use cases.

On this page:
Get Started
New Tutorials
Get Certified
Build Providers
All Tutorials

Additional Resources

- Recommended Pattern - TFE Backup
- Recommended Pattern - TFE Recovery & Restore
- Backup & Restore API
- Terraform Enterprise Releases
- Support KBase: Upgrading TFE
- Monitoring a Terraform Enterprise Instance
- Support KBase: Monitoring TFE
- TFE Log Forwarding
- TFE Grafana Dashboard

Need Additional Help?

Customer Success

Contact our Customer Success Management team with any questions. We will help coordinate the right resources for you to get your questions answered.

customer.success@hashicorp.com

Technical Support

Something not working quite right? Engage with HashiCorp Technical Support by opening a ticket for your issue at:

support.hashicorp.com

Discuss

Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers

discuss.hashicorp.com

Upcoming Webinars

Program Closing

We conclude the webinar series with a short recorded session

The session and accompanying materials include an Operational Readiness Checklist for Terraform Enterprise and links to all of the program materials and recordings

Additional Topics

Additional sessions are planned in both live and pre-recorded format

If you have topics you would like covered please share them with us

Action Items

- Establish Recovery Point Objectives (RPO) & Recovery Time Objectives (RTO) for all TFE instances (if not already established)
- Determine your organizations update cadence and create appropriate runbooks
- Determine telemetry and monitoring tools and patterns to be used with TFE instance(s)

Q&A

Thank you

customer.success@hashicorp.com

www.hashicorp.com/customer-success