Terraform Environment Setup

Status	Not Started

Complete Step-by-Step Guide for All Platforms

Document Version: 9.0

Date: October 2023

Audience: All Levels - Beginners to Enterprise Teams

Executive Summary: The Chef's Kitchen Setup

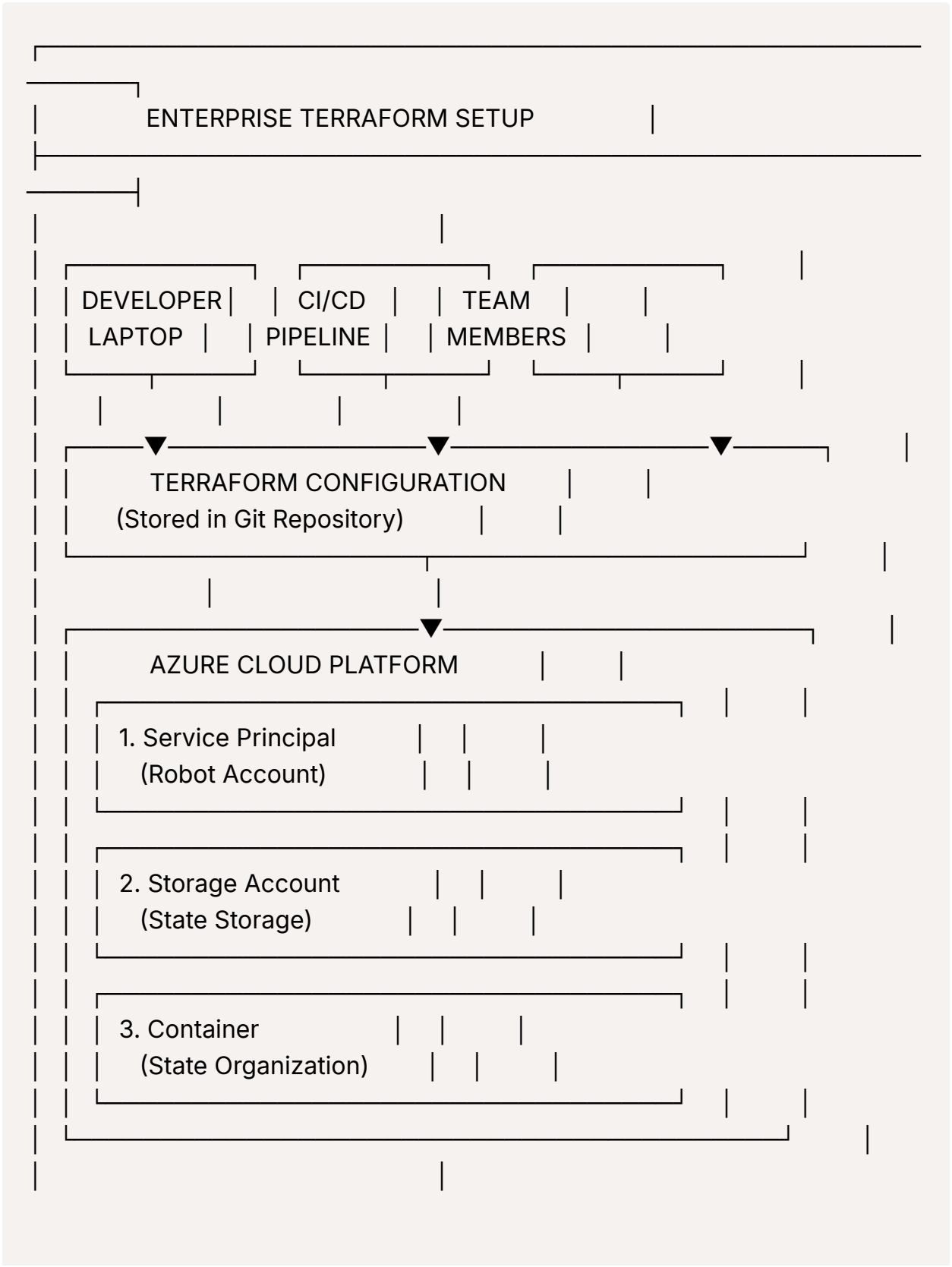
Think of Terraform Setup as Setting Up a Professional Kitchen:

Setup Component	Kitchen Analogy	Purpose
Terraform	Recipe Book & Cooking Tools	Infrastructure definition and deployment
Azure CLI	Kitchen Manager	Communication with Azure
Service Principal	Restaurant's Business License	Secure automated access
Storage Account	Recipe Card File Cabinet	Store Terraform state safely
Container	Recipe Card Drawer	Organized state storage
Backend	Kitchen Filing System	Where everything is stored

1. Complete Setup Overview

1.1 What We're Building

text


ENTERPRISE TERRAFORM SETUP
DEVELOPER LAPTOP
CI/CD PIPELINE
TEAM MEMBERS
TERRAFORM CONFIGURATION
(Stored in Git Repository)
AZURE CLOUD PLATFORM
1. Service Principal
(Robot Account)
2. Storage Account
(State Storage)
3. Container
(State Organization)

└──┘

1.2 Step-by-Step Roadmap

text

```
PHASE 1: LOCAL SETUP (Your Machine)
Step 1: Install Azure CLI
Step 2: Install Terraform
Step 3: Install Git

PHASE 2: AZURE AUTHENTICATION
Step 4: Login to Azure
Step 5: Create Service Principal
Step 6: Configure Permissions

PHASE 3: BACKEND INFRASTRUCTURE
Step 7: Create Resource Group
Step 8: Create Storage Account
Step 9: Create Container
Step 10: Configure Access

PHASE 4: TERRAFORM CONFIGURATION
Step 11: Create Terraform Project
Step 12: Configure Backend
Step 13: Test Setup

PHASE 5: ENTERPRISE READY
Step 14: Security Hardening
Step 15: CI/CD Integration
Step 16: Team Collaboration
```

2. Phase 1: Local Machine Setup

2.1 Step 1: Install Azure CLI

Azure CLI is the command-line interface for Microsoft Azure. It's like having a remote control for Azure in your terminal.

Windows Installation:

powershell

```
# Method 1: PowerShell (Recommended)
Invoke-WebRequest -Uri https://aka.ms/installazurecliwindows -OutFile .\Azur
eCLI.msi
Start-Process msiexec.exe -Wait -ArgumentList '/I AzureCLI.msi /quiet'
rm .\AzureCLI.msi

# Method 2: Microsoft Store (Simpler)
# Search for "Azure CLI" in Microsoft Store and install

# Method 3: Chocolatey (for developers)
choco install azure-cli

# Verify installation
az --version
```

macOS Installation:

bash

```
# Method 1: Homebrew (Recommended)
brew update && brew install azure-cli

# Method 2: Direct download
curl -L https://aka.ms/InstallAzureCli | bash

# For Apple Silicon (M1/M2/M3):
brew install azure-cli
arch -arm64 brew install azure-cli  # If on Rosetta
```

```
# Verify installation
az --version
```

Linux Installation (Ubuntu/Debian):

bash

```
# Method 1: Official Microsoft Repository
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

# Method 2: Manual installation
# 1. Get Microsoft signing key
curl -sL https://packages.microsoft.com/keys/microsoft.asc | \
    gpg --dearmor | \
    sudo tee /etc/apt/trusted.gpg.d/microsoft.gpg > /dev/null

# 2. Add Azure CLI repository
echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-cli/ $(l
sb_release -cs) main" | \
    sudo tee /etc/apt/sources.list.d/azure-cli.list

# 3. Install
sudo apt-get update
sudo apt-get install azure-cli

# Verify installation
az --version
```

Linux Installation (RHEL/CentOS/Fedora):

bash

```
# Method 1: Official Microsoft Repository
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

```
# RHEL 7/CentOS 7
echo -e "[azure-cli]
name=Azure CLI
baseurl=https://packages.microsoft.com/yumrepos/azure-cli
enabled=1
gpgcheck=1
gpgkey=https://packages.microsoft.com/keys/microsoft.asc" | sudo tee /etc/y
um.repos.d/azure-cli.repo

sudo yum install azure-cli

# RHEL 8/CentOS 8/Rocky Linux/AlmaLinux
sudo dnf install azure-cli

# Verify installation
az --version
```

Docker Installation (Alternative):

bash

```
# Run Azure CLI in Docker container
docker run -it --rm \
  -v ${HOME}/.azure:/root/.azure \
  -v ${PWD}:/workspace \
  mcr.microsoft.com/azure-cli:latest

# Create alias for easy use
echo "alias az='docker run -it --rm -v \${HOME}/.azure:/root/.azure -v \${PW
D}:/workspace mcr.microsoft.com/azure-cli:latest'" >> ~/.bashrc
source ~/.bashrc
```

2.2 Step 2: Install Terraform

Terraform is the Infrastructure as Code tool we'll use to define and deploy Azure resources.

Windows Installation:

powershell

```
# Method 1: Chocolatey (Recommended)
choco install terraform

# Method 2: Manual installation
# 1. Download from https://developer.hashicorp.com/terraform/downloads
# 2. Extract to C:\terraform
# 3. Add to PATH:
#    - Press Win + X, select System
#    - Advanced System Settings → Environment Variables
#    - Add C:\terraform to Path

# Method 3: PowerShell script
$TERRAFORM_VERSION="1.5.0"
$URL="https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/terraform_${TERRAFORM_VERSION}_windows_amd64.zip"
Invoke-WebRequest -Uri $URL -OutFile terraform.zip
Expand-Archive terraform.zip -DestinationPath C:\terraform\
[Environment]::SetEnvironmentVariable("Path", $env:Path + ";C:\terraform", [EnvironmentVariableTarget]::User)

# Verify installation
terraform --version
```

macOS Installation:

bash

```
# Method 1: Homebrew (Recommended)
brew tap hashicorp/tap
brew install hashicorp/tap/terraform

# Method 2: Manual installation
```

```
TERRAFORM_VERSION="1.5.0"
wget https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/terr
aform_${TERRAFORM_VERSION}_darwin_amd64.zip
unzip terraform_${TERRAFORM_VERSION}_darwin_amd64.zip
sudo mv terraform /usr/local/bin/

# For Apple Silicon (M1/M2/M3):
TERRAFORM_VERSION="1.5.0"
wget https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/terr
aform_${TERRAFORM_VERSION}_darwin_arm64.zip
unzip terraform_${TERRAFORM_VERSION}_darwin_arm64.zip
sudo mv terraform /usr/local/bin/

# Clean up
rm terraform_${TERRAFORM_VERSION}_*.zip

# Verify installation
terraform --version
```

Linux Installation:

bash

```
# Method 1: Using package manager

# Ubuntu/Debian
sudo apt-get update && sudo apt-get install -y gnupg software-properties-co
mmon
wget -O- https://apt.releases.hashicorp.com/gpg | \
    gpg --dearmor | \
    sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
    https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
    sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update
sudo apt install terraform
```

```
# RHEL/CentOS/Rocky Linux
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo
sudo yum -y install terraform

# Fedora
sudo dnf install -y dnf-plugins-core
sudo dnf config-manager --add-repo https://rpm.releases.hashicorp.com/fedora/hashicorp.repo
sudo dnf -y install terraform

# Method 2: Manual installation (any Linux)
TERRAFORM_VERSION="1.5.0"
wget https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/terraform_${TERRAFORM_VERSION}_linux_amd64.zip
unzip terraform_${TERRAFORM_VERSION}_linux_amd64.zip
sudo mv terraform /usr/local/bin/

# Verify installation
terraform --version
```

Automatic Installation Script (All Platforms):

bash

```
#!/bin/bash
# install-terraform.sh
# Automatic Terraform installation for all platforms

echo "=== AUTO TERRAFORM INSTALLER ==="
echo "Detecting platform..."

TERRAFORM_VERSION="1.5.0"
PLATFORM=""
```

```
ARCH=""

# Detect OS
case "$(uname -s)" in
   Linux*)     PLATFORM="linux";;
   Darwin*)    PLATFORM="darwin";;
   CYGWIN*)    PLATFORM="windows";;
   MINGW*)     PLATFORM="windows";;
   *)          PLATFORM="unknown"
esac

# Detect Architecture
case "$(uname -m)" in
   x86_64*)    ARCH="amd64";;
   arm64*)     ARCH="arm64";;
   aarch64*)   ARCH="arm64";;
   *)          ARCH="386"
esac

echo "Platform: $PLATFORM"
echo "Architecture: $ARCH"

if [ "$PLATFORM" = "unknown" ]; then
   echo "❌ Unsupported platform"
   exit 1
fi

# Download URL
URL="https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/ter
raform_${TERRAFORM_VERSION}_${PLATFORM}_${ARCH}.zip"

echo "Downloading Terraform from: $URL"

# Download and install
if command -v wget &> /dev/null; then
   wget $URL -O terraform.zip
```

```
elif command -v curl &> /dev/null; then
    curl -L $URL -o terraform.zip
else
    echo "❌ Need wget or curl to download"
    exit 1
fi

# Extract
echo "Extracting..."
unzip -o terraform.zip

# Install
echo "Installing..."
if [ "$PLATFORM" = "windows" ]; then
    # Windows
    mkdir -p C:\terraform
    mv terraform.exe C:\terraform\
    echo "Add C:\terraform to your PATH"
else
    # Linux/macOS
    sudo mv terraform /usr/local/bin/
fi

# Cleanup
rm -f terraform.zip

# Verify
echo ""
terraform --version
echo "✅ Terraform installed successfully!"
```

2.3 Step 3: Install Git (Version Control)

Git is essential for version controlling your Terraform code.

Windows Installation:

powershell

```
# Method 1: Git for Windows (Recommended)
# Download from https://git-scm.com/download/win
# Run installer with default options

# Method 2: Chocolatey
choco install git

# Method 3: Winget
winget install --id Git.Git -e --source winget

# Verify installation
git --version
```

macOS Installation:

bash

```
# Method 1: Homebrew (Recommended)
brew install git

# Method 2: Xcode Command Line Tools
xcode-select --install

# Verify installation
git --version
```

Linux Installation:

bash

```
# Ubuntu/Debian
sudo apt update && sudo apt install -y git

# RHEL/CentOS/Rocky Linux
```

```
sudo yum install -y git

# Fedora
sudo dnf install -y git

# Verify installation
git --version
```

2.4 Step 4: Verify Complete Installation

bash

```
#!/bin/bash
# verify-setup.sh
echo "=== COMPLETE SETUP VERIFICATION ==="
echo ""

echo "1. Azure CLI:"
if command -v az &> /dev/null; then
    az --version | head -1
    echo "✅ Azure CLI installed"
else
    echo "❌ Azure CLI NOT installed"
fi

echo ""
echo "2. Terraform:"
if command -v terraform &> /dev/null; then
    terraform --version | head -1
    echo "✅ Terraform installed"
else
    echo "❌ Terraform NOT installed"
fi

echo ""
echo "3. Git:"
```

```
if command -v git &> /dev/null; then
    git --version
    echo "✅ Git installed"
else
    echo "❌ Git NOT installed"
fi

echo ""
echo "4. Additional Tools:"

# Check for jq (JSON processor)
if command -v jq &> /dev/null; then
    echo "✅ jq installed"
else
    echo "⚠️  jq NOT installed (recommended)"
    echo "   Install: brew install jq (macOS) or apt install jq (Linux)"
fi

# Check for unzip
if command -v unzip &> /dev/null; then
    echo "✅ unzip installed"
else
    echo "⚠️  unzip NOT installed"
fi

echo ""
echo "=== SETUP STATUS ==="
if command -v az &> /dev/null && command -v terraform &> /dev/null && com
mand -v git &> /dev/null; then
    echo "✅ ALL REQUIRED TOOLS INSTALLED"
    echo "Ready for Phase 2: Azure Authentication"
else
    echo "❌ MISSING SOME TOOLS"
    echo "Please install missing tools above"
fi
```

3. Phase 2: Azure Authentication Setup

3.1 Step 5: Login to Azure

There are multiple authentication methods. Choose based on your needs:

Method A: Interactive Login (For Developers)

bash

```
# This opens a browser for authentication
az login

# If you have multiple subscriptions, select one
az account list --output table

# Set active subscription
az account set --subscription "Your-Subscription-Name"

# Verify login
az account show --output table
```

Method B: Service Principal Login (For Automation)

bash

```
# First create service principal (we'll do this next)
# Then login with it
az login --service-principal \
    --username $ARM_CLIENT_ID \
    --password $ARM_CLIENT_SECRET \
    --tenant $ARM_TENANT_ID
```

Method C: Managed Identity (For Azure Resources)

bash

```
# For VMs, Azure Functions, etc. with Managed Identity
az login --identity

# For user-assigned identity
az login --identity --username $CLIENT_ID
```

Method D: Device Code (For Headless Systems)

bash

```
# For systems without browser (SSH sessions, containers)
az login --use-device-code

# You'll get a code to enter at https://microsoft.com/devicelogin
```

3.2 Step 6: Create Service Principal

What is a Service Principal? Think of it as a **robot user account** for automation. It's like creating a dedicated employee (robot) that only does Terraform deployments.

Option 1: Basic Service Principal (Quick Start)

bash

```
#!/bin/bash
# create-sp-basic.sh
echo "=== CREATING BASIC SERVICE PRINCIPAL ==="

# Get subscription ID
SUBSCRIPTION_ID=$(az account show --query id -o tsv)
echo "Subscription ID: $SUBSCRIPTION_ID"

# Create Service Principal with Contributor role
az ad sp create-for-rbac \
    --name "terraform-sp" \
```

```
    --role "Contributor" \
    --scopes "/subscriptions/$SUBSCRIPTION_ID" \
    --years 2

# Output will look like:
# {
#   "appId": "00000000-0000-0000-0000-000000000000",
#   "displayName": "terraform-sp",
#   "password": "abc123...",
#   "tenant": "00000000-0000-0000-0000-000000000000"
# }

echo ""
echo "✅ Service Principal created!"
echo "Save these credentials securely!"
```

Option 2: Advanced Service Principal with Specific Permissions

bash

```
#!/bin/bash
# create-sp-advanced.sh
echo "=== CREATING ADVANCED SERVICE PRINCIPAL ==="

# Get subscription ID and tenant ID
SUBSCRIPTION_ID=$(az account show --query id -o tsv)
TENANT_ID=$(az account show --query tenantId -o tsv)

echo "Subscription: $SUBSCRIPTION_ID"
echo "Tenant: $TENANT_ID"

# 1. Create Azure AD Application
APP_NAME="terraform-enterprise-sp"
APP_ID=$(az ad app create \
    --display-name $APP_NAME \
    --query appId -o tsv)
```

```
echo "Application ID: $APP_ID"

# 2. Create Service Principal
SP_ID=$(az ad sp create --id $APP_ID --query id -o tsv)
echo "Service Principal ID: $SP_ID"

# 3. Create Client Secret (Password)
SECRET=$(az ad app credential reset \
    --id $APP_ID \
    --years 2 \
    --query password -o tsv)

echo "Client Secret: $SECRET"

# 4. Assign roles with specific scopes (least privilege)
# Create resource group for Terraform state first
az group create --name terraform-state-rg --location eastus

# Assign Storage Account Contributor for state management
az role assignment create \
    --assignee $APP_ID \
    --role "Storage Account Contributor" \
    --scope "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/terraform-sta
te-rg"

# Assign Contributor for specific resource groups only
az role assignment create \
    --assignee $APP_ID \
    --role "Contributor" \
    --scope "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/prod-*"

az role assignment create \
    --assignee $APP_ID \
    --role "Contributor" \
    --scope "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/dev-*"
```

```
# 5. Output credentials
echo ""
echo "=== SERVICE PRINCIPAL CREDENTIALS ==="
echo "ARM_CLIENT_ID: $APP_ID"
echo "ARM_CLIENT_SECRET: $SECRET"
echo "ARM_TENANT_ID: $TENANT_ID"
echo "ARM_SUBSCRIPTION_ID: $SUBSCRIPTION_ID"
echo ""
echo "✅ Advanced Service Principal created with least privilege!"
```

Option 3: Enterprise-Grade Service Principal with Custom Role

bash

```
#!/bin/bash
# create-sp-enterprise.sh
echo "=== ENTERPRISE SERVICE PRINCIPAL ==="

SUBSCRIPTION_ID=$(az account show --query id -o tsv)
TENANT_ID=$(az account show --query tenantId -o tsv)

# 1. Create custom RBAC role for Terraform
cat > terraform-custom-role.json << EOF
{
    "Name": "Terraform Deployer",
    "Description": "Custom role for Terraform deployments with least privilege",
    "Actions": [
        "Microsoft.Resources/subscriptions/resourceGroups/read",
        "Microsoft.Resources/subscriptions/resourceGroups/write",
        "Microsoft.Resources/subscriptions/resourceGroups/delete",
        "Microsoft.Resources/deployments/*",
        "Microsoft.Authorization/*/read",
        "Microsoft.Insights/alertRules/*",
        "Microsoft.Support/*"
    ],
```

```
    "NotActions": [],
    "AssignableScopes": [
      "/subscriptions/$SUBSCRIPTION_ID"
    ]
}
EOF

# Create custom role
az role definition create --role-definition terraform-custom-role.json

# 2. Create Service Principal
SP_CREDENTIALS=$(az ad sp create-for-rbac \
    --name "terraform-enterprise" \
    --role "Terraform Deployer" \
    --scopes "/subscriptions/$SUBSCRIPTION_ID" \
    --years 1 \
    --create-cert)

# 3. Save credentials to environment file
cat > .env << EOF
# Terraform Service Principal Credentials
export ARM_CLIENT_ID=$(echo $SP_CREDENTIALS | jq -r '.appId')
export ARM_CLIENT_SECRET=$(echo $SP_CREDENTIALS | jq -r '.password')
export ARM_TENANT_ID=$(echo $SP_CREDENTIALS | jq -r '.tenant')
export ARM_SUBSCRIPTION_ID=$SUBSCRIPTION_ID

# For certificate authentication (alternative)
# export ARM_CLIENT_CERTIFICATE_PATH=/path/to/cert.pfx
# export ARM_CLIENT_CERTIFICATE_PASSWORD=your_password
EOF

# 4. Also save to Azure Key Vault (secure storage)
az keyvault create --name "kv-terraform-secrets" \
    --resource-group "security-rg" \
    --location eastus
```

```
az keyvault secret set --vault-name "kv-terraform-secrets" \
    --name "terraform-client-id" \
    --value $(echo $SP_CREDENTIALS | jq -r '.appId')

az keyvault secret set --vault-name "kv-terraform-secrets" \
    --name "terraform-client-secret" \
    --value $(echo $SP_CREDENTIALS | jq -r '.password')

echo ""
echo "✅ Enterprise Service Principal created!"
echo "Credentials saved to .env and Azure Key Vault"
```

3.3 Step 7: Configure Environment Variables

bash

```
#!/bin/bash
# configure-environment.sh
echo "=== CONFIGURING TERRAFORM ENVIRONMENT ==="

# Load credentials from .env if exists
if [ -f .env ]; then
    source .env
    echo "Loaded credentials from .env"
else
    # Or set them manually
    read -p "Enter ARM_CLIENT_ID: " ARM_CLIENT_ID
    read -sp "Enter ARM_CLIENT_SECRET: " ARM_CLIENT_SECRET
    echo
    read -p "Enter ARM_TENANT_ID: " ARM_TENANT_ID
    read -p "Enter ARM_SUBSCRIPTION_ID: " ARM_SUBSCRIPTION_ID
fi

# Set environment variables
export ARM_CLIENT_ID=$ARM_CLIENT_ID
export ARM_CLIENT_SECRET=$ARM_CLIENT_SECRET
```

```
export ARM_TENANT_ID=$ARM_TENANT_ID
export ARM_SUBSCRIPTION_ID=$ARM_SUBSCRIPTION_ID

# Optional: For specific features
export ARM_SKIP_PROVIDER_REGISTRATION=true
export ARM_USE_MSI=false
export ARM_USE_AZUREAD=false

# Save to shell profile
echo ""
echo "Saving to shell profile..."

SHELL_PROFILE=""
if [ -n "$BASH_VERSION" ]; then
    SHELL_PROFILE="$HOME/.bashrc"
elif [ -n "$ZSH_VERSION" ]; then
    SHELL_PROFILE="$HOME/.zshrc"
fi

if [ -n "$SHELL_PROFILE" ]; then
    cat >> $SHELL_PROFILE << EOF

# Terraform Azure Configuration
export ARM_CLIENT_ID="$ARM_CLIENT_ID"
export ARM_CLIENT_SECRET="$ARM_CLIENT_SECRET"
export ARM_TENANT_ID="$ARM_TENANT_ID"
export ARM_SUBSCRIPTION_ID="$ARM_SUBSCRIPTION_ID"
EOF
    echo "Added to $SHELL_PROFILE"
fi

# Test authentication
echo ""
echo "Testing authentication..."
az login --service-principal \
    --username $ARM_CLIENT_ID \
```

```
    --password $ARM_CLIENT_SECRET \
    --tenant $ARM_TENANT_ID

if [ $? -eq 0 ]; then
    echo "✅ Authentication successful!"
    az account show --query "{Subscription:name, Tenant:tenantId}"
else
    echo "❌ Authentication failed"
    exit 1
fi
```

4. Phase 3: Backend Infrastructure Setup

4.1 Step 8: Create Resource Group

Why a separate Resource Group? Isolate Terraform state management from application resources.

bash

```
#!/bin/bash
# create-resource-group.sh
echo "=== CREATING RESOURCE GROUP FOR TERRAFORM STATE ==="

# Configuration
RESOURCE_GROUP="terraform-state-rg"
LOCATION="eastus"
TAGS="Environment=Management ManagedBy=Terraform Purpose=StateStor
age"

echo "Resource Group: $RESOURCE_GROUP"
echo "Location: $LOCATION"

# Create Resource Group
az group create \
    --name $RESOURCE_GROUP \
```

```
    --location $LOCATION \
    --tags $TAGS

# Verify creation
az group show --name $RESOURCE_GROUP --query "{Name:name, Location:l
ocation, Tags:tags}"

echo ""
echo "✅ Resource Group created successfully!"
```

4.2 Step 9: Create Storage Account

Storage Account Requirements for Terraform State:

1. **Standard LRS** (Locally Redundant Storage) is sufficient
2. **HTTPS** enabled by default
3. **Versioning** enabled (recommended)
4. **Soft delete** enabled (recommended)
5. **Public access** disabled (security)

bash

```
#!/bin/bash
# create-storage-account.sh
echo "=== CREATING STORAGE ACCOUNT FOR TERRAFORM STATE ==="

# Configuration
RESOURCE_GROUP="terraform-state-rg"
LOCATION="eastus"

# Generate unique storage account name (3-24 chars, lowercase letters and n
umbers)
STORAGE_ACCOUNT_NAME="tfstate$(openssl rand -hex 4)"
CONTAINER_NAME="tfstate"
SKU="Standard_LRS"
```

```
KIND="StorageV2"

echo "Storage Account: $STORAGE_ACCOUNT_NAME"
echo "Container: $CONTAINER_NAME"
echo "SKU: $SKU"

# Create Storage Account
az storage account create \
    --name $STORAGE_ACCOUNT_NAME \
    --resource-group $RESOURCE_GROUP \
    --location $LOCATION \
    --sku $SKU \
    --kind $KIND \
    --encryption-services blob \
    --min-tls-version TLS1_2 \
    --allow-blob-public-access false \
    --default-action Deny

# Enable advanced features
echo "Enabling advanced features..."

# Enable versioning (track changes to state files)
az storage account blob-service-properties update \
    --account-name $STORAGE_ACCOUNT_NAME \
    --enable-versioning true

# Enable soft delete (recover deleted blobs)
az storage account blob-service-properties update \
    --account-name $STORAGE_ACCOUNT_NAME \
    --enable-delete-retention true \
    --delete-retention-days 30

# Enable change feed (audit trail)
az storage account blob-service-properties update \
    --account-name $STORAGE_ACCOUNT_NAME \
    --enable-change-feed true \
```

```
    --change-feed-retention-days 30

# Get storage account key (for authentication)
ACCOUNT_KEY=$(az storage account keys list \
    --resource-group $RESOURCE_GROUP \
    --account-name $STORAGE_ACCOUNT_NAME \
    --query '[0].value' -o tsv)

echo ""
echo "✅ Storage Account created successfully!"
echo ""
echo "=== STORAGE ACCOUNT DETAILS ==="
echo "Name: $STORAGE_ACCOUNT_NAME"
echo "Resource Group: $RESOURCE_GROUP"
echo "Primary Key: $ACCOUNT_KEY"
echo ""
echo "⚠️  Save these credentials securely!"
```

4.3 Step 10: Create Container

Containers organize state files (like folders in a file system).

bash

```
#!/bin/bash
# create-container.sh
echo "=== CREATING BLOB CONTAINER FOR TERRAFORM STATE ==="

# Configuration (use values from previous steps)
RESOURCE_GROUP="terraform-state-rg"
STORAGE_ACCOUNT_NAME="tfstateXXXX"  # Replace with your storage acc
ount
CONTAINER_NAME="tfstate"

echo "Storage Account: $STORAGE_ACCOUNT_NAME"
echo "Container: $CONTAINER_NAME"
```

```
# Create container with private access
az storage container create \
    --name $CONTAINER_NAME \
    --account-name $STORAGE_ACCOUNT_NAME \
    --auth-mode key \
    --public-access off

# Create additional containers for different environments
ENVIRONMENTS=("dev" "staging" "prod" "shared")

for ENV in "${ENVIRONMENTS[@]}"; do
    echo "Creating container for $ENV environment..."
    az storage container create \
        --name "tfstate-$ENV" \
        --account-name $STORAGE_ACCOUNT_NAME \
        --auth-mode key \
        --public-access off \
        --metadata "environment=$ENV" "managedby=terraform"
done

# List all containers
echo ""
echo "Listing all containers..."
az storage container list \
    --account-name $STORAGE_ACCOUNT_NAME \
    --auth-mode key \
    --query "[].name" -o table

echo ""
echo "✅ Containers created successfully!"
```

4.4 Step 11: Configure Storage Account Security

bash

```
#!/bin/bash
# configure-storage-security.sh
echo "=== CONFIGURING STORAGE ACCOUNT SECURITY ==="

RESOURCE_GROUP="terraform-state-rg"
STORAGE_ACCOUNT_NAME="tfstateXXXX"  # Your storage account

# 1. Enable Azure Defender for Storage
az security pricing create \
    --name "StorageAccounts" \
    --tier "Standard"

# 2. Configure Network Rules (allow only specific IPs/VNets)
# Get your public IP
MY_IP=$(curl -s ifconfig.me)

az storage account network-rule add \
    --resource-group $RESOURCE_GROUP \
    --account-name $STORAGE_ACCOUNT_NAME \
    --ip-address "$MY_IP"

# 3. Enable Storage Service Encryption (SSE)
az storage account update \
    --name $STORAGE_ACCOUNT_NAME \
    --resource-group $RESOURCE_GROUP \
    --encryption-key-type-for-table Service \
    --encryption-key-type-for-queue Service

# 4. Configure CORS (if accessing via web)
az storage cors add \
    --account-name $STORAGE_ACCOUNT_NAME \
    --services b \
    --origins "*" \
    --methods GET PUT OPTIONS \
    --max-age 3600 \
```

```
    --allowed-headers "*"

# 5. Set up diagnostic settings
LOG_ANALYTICS_WORKSPACE=$(az monitor log-analytics workspace list \
    --resource-group $RESOURCE_GROUP \
    --query "[0].id" -o tsv)

if [ -n "$LOG_ANALYTICS_WORKSPACE" ]; then
    az monitor diagnostic-settings create \
        --resource $(az storage account show \
            --name $STORAGE_ACCOUNT_NAME \
            --resource-group $RESOURCE_GROUP \
            --query id -o tsv) \
        --name "StorageDiagnostics" \
        --workspace $LOG_ANALYTICS_WORKSPACE \
        --logs '[{"category": "StorageRead", "enabled": true}, {"category": "Stora
geWrite", "enabled": true}, {"category": "StorageDelete", "enabled": true}]' \
        --metrics '[{"category": "Transaction", "enabled": true}]'
fi

echo ""
echo "✅ Storage account security configured!"
```

5. Phase 4: Terraform Configuration

5.1 Step 12: Create Terraform Project Structure

bash

```
#!/bin/bash
# create-terraform-project.sh
echo "=== CREATING TERRAFORM PROJECT STRUCTURE ==="

PROJECT_NAME="my-terraform-project"
echo "Project Name: $PROJECT_NAME"
```

```
# Create directory structure
mkdir -p $PROJECT_NAME/{environments,modules,scripts,tests}
cd $PROJECT_NAME

# Create essential files
touch main.tf variables.tf outputs.tf providers.tf terraform.tfvars .gitignore REA
DME.md

# Create environment-specific files
for ENV in dev staging prod; do
    mkdir -p environments/$ENV
    touch environments/$ENV/main.tf environments/$ENV/variables.tf environ
ments/$ENV/terraform.tfvars
done

# Create module structure
mkdir -p modules/{network,compute,storage,database,security}
for MODULE in network compute storage database security; do
    touch modules/$MODULE/{main.tf,variables.tf,outputs.tf,README.md}
done

# Create scripts
cat > scripts/setup.sh << 'EOF'
#!/bin/bash
# Setup script for Terraform project

echo "=== PROJECT SETUP ==="

# Initialize Terraform
terraform init

# Validate configuration
terraform validate

# Plan deployment
```

```
terraform plan -out=tfplan

echo "Setup complete!"
EOF

chmod +x scripts/setup.sh

# Create .gitignore for Terraform
cat > .gitignore << 'EOF'
# Local .terraform directories
**/.terraform/*

# .tfstate files
*.tfstate
*.tfstate.*

# Crash log files
crash.log
crash.*.log

# Exclude all .tfvars files, which are likely to contain sensitive data
*.tfvars
*.tfvars.json

# Ignore override files as they are usually used to override resources locally
*_override.tf
*_override.tf.json

# Include override files you do wish to add to version control using negated p
attern
# !example_override.tf

# Include tfplan files to ignore the plan output of command: terraform plan -ou
t=tfplan
# example: *tfplan*
```

```
# Ignore CLI configuration files
.terraformrc
terraform.rc

# IDE files
.vscode/
.idea/
*.swp
*.swo

# Environment files
.env
*.env
EOF

# Create README.md
cat > README.md << 'EOF'
# Terraform Project

## Overview
This project manages infrastructure using Terraform.

## Structure
- `environments/` - Environment-specific configurations
- `modules/` - Reusable Terraform modules
- `scripts/` - Helper scripts
- `tests/` - Test configurations

## Setup
1. Run `scripts/setup.sh`
2. Configure backend in `backend.tf`
3. Set variables in `terraform.tfvars`

## Usage
```bash
# Initialize
```
```

```
terraform init

# Plan
terraform plan

# Apply
terraform apply

# Destroy
terraform destroy
```

EOF

echo ""

echo "✅ Project structure created!"

echo ""

tree .

text

```
### **5.2 Step 13: Configure Terraform Backend**

```bash
# Create backend configuration
cat > backend.tf << 'EOF'
terraform {
  backend "azurerm" {
    resource_group_name  = "terraform-state-rg"
    storage_account_name = "tfstateXXXX"  # Replace with your storage accou
nt
    container_name       = "tfstate"
    key                  = "terraform.tfstate"

    # Optional: Use SAS token for authentication
    # sas_token = var.sas_token
```
```

```
    # Optional: Use access key (not recommended for production)
    # access_key = var.access_key

    # Optional: Use Managed Service Identity
    # use_msi = true

    # Optional: Use Azure AD authentication
    # use_azuread_auth = true
  }
}
EOF

# Create provider configuration
cat > providers.tf << 'EOF'
terraform {
  required_version = ">= 1.0.0"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }

    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }

    time = {
      source  = "hashicorp/time"
      version = "~> 0.9.0"
    }
  }
}

provider "azurerm" {
```

```
  features {
    resource_group {
      prevent_deletion_if_contains_resources = true
    }

    key_vault {
      purge_soft_delete_on_destroy = true
    }

    virtual_machine {
      delete_os_disk_on_deletion = true
    }
  }

  # Authentication via environment variables
  # ARM_CLIENT_ID
  # ARM_CLIENT_SECRET
  # ARM_TENANT_ID
  # ARM_SUBSCRIPTION_ID
}

provider "random" {}

provider "time" {}
EOF
```

5.3 Step 14: Create Initial Terraform Configuration

bash

```
# Create main configuration
cat > main.tf << 'EOF'
# main.tf - Initial Terraform configuration

# Create resource group for our resources
resource "azurerm_resource_group" "main" {
```

```
  name     = "rg-${var.environment}-${var.project_name}"
  location = var.location

  tags = merge(var.tags, {
    Environment = var.environment
    ManagedBy   = "Terraform"
    Deployment  = formatdate("YYYY-MM-DD", timestamp())
  })
}

# Create storage account for application data
resource "azurerm_storage_account" "app" {
  name                     = "st${var.environment}${var.project_name}${random_stri
ng.suffix.result}"
  resource_group_name      = azurerm_resource_group.main.name
  location                 = azurerm_resource_group.main.location
  account_tier             = "Standard"
  account_replication_type = "LRS"

  tags = azurerm_resource_group.main.tags
}

# Random string for unique names
resource "random_string" "suffix" {
  length  = 8
  special = false
  upper   = false
}

# Output values
output "resource_group_name" {
  value       = azurerm_resource_group.main.name
  description = "Name of the resource group"
}

output "storage_account_name" {
```

```
  value       = azurerm_storage_account.app.name
  description = "Name of the storage account"
  sensitive   = false
}
EOF

# Create variables
cat > variables.tf << 'EOF'
variable "environment" {
  type        = string
  description = "Environment name (dev, staging, prod)"
  default     = "dev"

  validation {
    condition     = contains(["dev", "staging", "prod"], var.environment)
    error_message = "Environment must be dev, staging, or prod."
  }
}

variable "project_name" {
  type        = string
  description = "Name of the project"
  default     = "myproject"
}

variable "location" {
  type        = string
  description = "Azure region"
  default     = "eastus"
}

variable "tags" {
  type        = map(string)
  description = "Tags to apply to all resources"
  default     = {}
}
```

```
EOF

# Create terraform.tfvars
cat > terraform.tfvars << 'EOF'
# Development environment variables
environment = "dev"
project_name = "terraformdemo"
location = "eastus"

tags = {
  CostCenter = "IT"
  Department = "Engineering"
}
EOF
```

5.4 Step 15: Test the Setup

bash

```
#!/bin/bash
# test-setup.sh
echo "=== TESTING TERRAFORM SETUP ==="

echo "1. Checking environment variables..."
env | grep ARM_

echo ""
echo "2. Testing Azure authentication..."
az account show --query "{Name:name, User:user.name}" -o table

echo ""
echo "3. Initializing Terraform..."
terraform init

echo ""
echo "4. Validating configuration..."
```

```
terraform validate

echo ""
echo "5. Planning deployment..."
terraform plan -out=tfplan

echo ""
echo "6. Checking plan output..."
terraform show -json tfplan | jq -r '.planned_values.outputs'

echo ""
read -p "7. Apply the plan? (yes/no): " apply_confirm

if [ "$apply_confirm" = "yes" ]; then
    echo "Applying..."
    terraform apply tfplan

    echo ""
    echo "8. Checking outputs..."
    terraform output

    echo ""
    echo "✅ Setup test completed successfully!"
else
    echo "Skipping apply. Cleanup..."
    rm -f tfplan
    echo "✅ Setup test completed (without apply)"
fi

echo ""
echo "=== NEXT STEPS ==="
echo "1. Review the created resources in Azure Portal"
echo "2. Check state file in storage account"
echo "3. Try modifying variables and re-running"
echo "4. Add more resources to main.tf"
```

6. Phase 5: Enterprise-Ready Enhancements

6.1 Step 16: Security Hardening

bash

```
#!/bin/bash
# enterprise-security.sh
echo "=== ENTERPRISE SECURITY SETUP ==="

# 1. Create Key Vault for secrets
az keyvault create \
    --name "kv-$(openssl rand -hex 4)" \
    --resource-group "terraform-state-rg" \
    --location eastus \
    --sku standard \
    --enable-rbac-authorization true

# 2. Store Terraform credentials in Key Vault
KEYVAULT_NAME=$(az keyvault list --resource-group "terraform-state-rg" --
query "[0].name" -o tsv)

az keyvault secret set \
    --vault-name $KEYVAULT_NAME \
    --name "terraform-client-id" \
    --value $ARM_CLIENT_ID

az keyvault secret set \
    --vault-name $KEYVAULT_NAME \
    --name "terraform-client-secret" \
    --value $ARM_CLIENT_SECRET

az keyvault secret set \
    --vault-name $KEYVAULT_NAME \
    --name "storage-account-key" \
```

```
    --value $ACCOUNT_KEY

# 3. Enable Azure Policy for compliance
az policy assignment create \
    --name "require-terraform-tags" \
    --display-name "Require tags on Terraform resources" \
    --policy "/providers/Microsoft.Authorization/policyDefinitions/1e30110a-5ceb-460c-a204-c1c3969c6d62" \
    --params '{"tagName": {"value": "ManagedBy"}, "tagValue": {"value": "Terraform"}}' \
    --scope "/subscriptions/$ARM_SUBSCRIPTION_ID"

# 4. Set up Azure Monitor alerts
az monitor action-group create \
    --name "TerraformAlerts" \
    --resource-group "terraform-state-rg" \
    --short-name "tf-alerts"

# 5. Enable Azure Defender for Cloud
az security pricing create \
    --name "VirtualMachines" \
    --tier "Standard"

echo ""
echo "✅ Enterprise security configured!"
```

6.2 Step 17: CI/CD Integration Setup

yaml

```
# Create GitHub Actions workflow
mkdir -p .github/workflows
cat > .github/workflows/terraform.yml << 'EOF'
name: 'Terraform CI/CD'

on:
```

```
  push:
    branches: [ main, dev ]
  pull_request:
    branches: [ main ]

env:
  ARM_CLIENT_ID: ${{ secrets.ARM_CLIENT_ID }}
  ARM_CLIENT_SECRET: ${{ secrets.ARM_CLIENT_SECRET }}
  ARM_SUBSCRIPTION_ID: ${{ secrets.ARM_SUBSCRIPTION_ID }}
  ARM_TENANT_ID: ${{ secrets.ARM_TENANT_ID }}
  TERRAFORM_VERSION: '1.5.0'

jobs:
  terraform:
    name: 'Terraform'
    runs-on: ubuntu-latest

    defaults:
      run:
        shell: bash

    steps:
    - name: Checkout
      uses: actions/checkout@v3

    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v2
      with:
        terraform_version: ${{ env.TERRAFORM_VERSION }}
        terraform_wrapper: false

    - name: Terraform Init
      run: terraform init

    - name: Terraform Format
      run: terraform fmt -check -recursive
```

```
    - name: Terraform Validate
      run: terraform validate

    - name: Terraform Security Scan
      uses: aquasecurity/tfsec-action@master
      with:
        working_directory: '.'

    - name: Terraform Plan
      run: terraform plan -out=tfplan

    - name: Upload Plan Artifact
      uses: actions/upload-artifact@v3
      with:
        name: tfplan
        path: tfplan

    - name: Terraform Apply
      if: github.ref == 'refs/heads/main' && github.event_name == 'push'
      run: terraform apply -auto-approve tfplan
EOF
```

6.3 Step 18: Team Collaboration Setup

bash

```
#!/bin/bash
# team-collaboration.sh
echo "=== TEAM COLLABORATION SETUP ==="

# 1. Create GitHub repository
read -p "GitHub username: " GITHUB_USERNAME
read -p "Repository name: " REPO_NAME

# Create repo (requires GitHub CLI or token)
```

```
gh repo create $REPO_NAME --public --description "Terraform configuration"

# 2. Initialize git in project
git init
git add .
git commit -m "Initial Terraform configuration"

# 3. Add remote and push
git remote add origin https://github.com/$GITHUB_USERNAME/$REPO_NAME.
git
git branch -M main
git push -u origin main

# 4. Create branch protection rules
# Requires GitHub API token
read -p "GitHub token: " GITHUB_TOKEN

curl -X PUT \
  -H "Authorization: token $GITHUB_TOKEN" \
  -H "Accept: application/vnd.github.v3+json" \
  https://api.github.com/repos/$GITHUB_USERNAME/$REPO_NAME/branches/
main/protection \
  -d '{
    "required_status_checks": {
      "strict": true,
      "contexts": ["terraform-validate", "terraform-plan"]
    },
    "enforce_admins": true,
    "required_pull_request_reviews": {
      "dismiss_stale_reviews": true,
      "require_code_owner_reviews": false,
      "required_approving_review_count": 1
    },
    "restrictions": null
  }'
```

```
# 5. Create team structure
cat > TEAMS.md << 'EOF'
# Team Structure

## Infrastructure Team
- Role: Manage Terraform configurations
- Permissions: Write access to main branch
- Responsibilities:
  * Review Terraform changes
  * Approve deployments
  * Manage state files

## Development Team
- Role: Use infrastructure
- Permissions: Read access
- Responsibilities:
  * Request infrastructure changes via PRs
  * Test in development environment

## Security Team
- Role: Security compliance
- Permissions: Read access + security scanning
- Responsibilities:
  * Review security configurations
  * Monitor compliance
EOF

echo ""
echo "✅ Team collaboration setup complete!"
echo ""
echo "Next:"
echo "1. Add team members to GitHub"
echo "2. Set up branch permissions"
echo "3. Create pull request templates"
```

7. Complete Automation Script

7.1 One-Click Setup Script

bash

```
#!/bin/bash
# terraform-enterprise-setup.sh
# Complete Terraform setup script

set -e  # Exit on error

echo "================================================"
echo "    TERRAFORM ENTERPRISE SETUP WIZARD"
echo "================================================"
echo ""

# Configuration
read -p "Enter project name: " PROJECT_NAME
read -p "Enter Azure region (default: eastus): " REGION
REGION=${REGION:-eastus}

echo ""
echo "Starting setup for: $PROJECT_NAME in $REGION"
echo ""

# Phase 1: Prerequisites Check
echo "=== PHASE 1: CHECKING PREREQUISITES ==="
check_prerequisites() {
    echo "Checking for required tools..."

    local missing=0

    # Check Azure CLI
    if ! command -v az &> /dev/null; then
        echo "❌ Azure CLI not found"
```

```
        missing=1
    else
        echo "✅ Azure CLI installed"
    fi

    # Check Terraform
    if ! command -v terraform &> /dev/null; then
        echo "❌ Terraform not found"
        missing=1
    else
        echo "✅ Terraform installed"
    fi

    # Check Git
    if ! command -v git &> /dev/null; then
        echo "❌ Git not found"
        missing=1
    else
        echo "✅ Git installed"
    fi

    # Check jq
    if ! command -v jq &> /dev/null; then
        echo "⚠️  jq not found (recommended)"
    else
        echo "✅ jq installed"
    fi

    if [ $missing -eq 1 ]; then
        echo ""
        echo "❌ Missing required tools. Please install them first."
        exit 1
    fi

    echo "✅ All prerequisites satisfied"
}
```

```
check_prerequisites

# Phase 2: Azure Authentication
echo ""
echo "=== PHASE 2: AZURE AUTHENTICATION ==="
authenticate_azure() {
    echo "Logging into Azure..."

    # Try interactive login
    az login

    # Get subscription
    SUBSCRIPTIONS=$(az account list --query "[].{Name:name, Id:id}" -o tabl
e)
    echo ""
    echo "Available subscriptions:"
    echo "$SUBSCRIPTIONS"

    read -p "Enter subscription ID to use: " SUBSCRIPTION_ID
    az account set --subscription $SUBSCRIPTION_ID

    TENANT_ID=$(az account show --query tenantId -o tsv)

    echo "✅ Authenticated to subscription: $SUBSCRIPTION_ID"
}

authenticate_azure

# Phase 3: Create Service Principal
echo ""
echo "=== PHASE 3: CREATING SERVICE PRINCIPAL ==="
create_service_principal() {
    echo "Creating Service Principal..."

    SP_NAME="terraform-$PROJECT_NAME"
```

```
    SP_JSON=$(az ad sp create-for-rbac \
        --name $SP_NAME \
        --role Contributor \
        --scopes "/subscriptions/$SUBSCRIPTION_ID" \
        --years 2 \
        --output json)

    ARM_CLIENT_ID=$(echo $SP_JSON | jq -r '.appId')
    ARM_CLIENT_SECRET=$(echo $SP_JSON | jq -r '.password')
    ARM_TENANT_ID=$(echo $SP_JSON | jq -r '.tenant')

    # Save to .env file
    cat > .env << EOF
# Terraform Service Principal
export ARM_CLIENT_ID="$ARM_CLIENT_ID"
export ARM_CLIENT_SECRET="$ARM_CLIENT_SECRET"
export ARM_TENANT_ID="$ARM_TENANT_ID"
export ARM_SUBSCRIPTION_ID="$SUBSCRIPTION_ID"
EOF

    source .env

    echo "✅ Service Principal created: $SP_NAME"
    echo "Credentials saved to .env file"
}

create_service_principal

# Phase 4: Create Backend Infrastructure
echo ""
echo "=== PHASE 4: CREATING BACKEND INFRASTRUCTURE ==="
create_backend() {
    echo "Creating backend resources..."

    # Resource Group
```

```
    RG_NAME="terraform-state-$PROJECT_NAME"
    echo "Creating Resource Group: $RG_NAME"
    az group create --name $RG_NAME --location $REGION

    # Storage Account
    STORAGE_NAME="tfstate${PROJECT_NAME}${RANDOM}"
    STORAGE_NAME=$(echo "$STORAGE_NAME" | tr '[:upper:]' '[:lower:]' | tr -
cd 'a-z0-9')
    STORAGE_NAME=${STORAGE_NAME:0:24}

    echo "Creating Storage Account: $STORAGE_NAME"
    az storage account create \
        --name $STORAGE_NAME \
        --resource-group $RG_NAME \
        --location $REGION \
        --sku Standard_LRS \
        --encryption-services blob \
        --min-tls-version TLS1_2 \
        --allow-blob-public-access false

    # Container
    CONTAINER_NAME="tfstate"
    echo "Creating Container: $CONTAINER_NAME"
    az storage container create \
        --name $CONTAINER_NAME \
        --account-name $STORAGE_NAME \
        --auth-mode key

    # Get storage key
    STORAGE_KEY=$(az storage account keys list \
        --resource-group $RG_NAME \
        --account-name $STORAGE_NAME \
        --query '[0].value' -o tsv)

    # Save backend info
    cat > backend.info << EOF
```

```
# Terraform Backend Configuration
resource_group_name  = "$RG_NAME"
storage_account_name = "$STORAGE_NAME"
container_name       = "$CONTAINER_NAME"
key                  = "terraform.tfstate"

# Storage Account Key (for reference)
storage_key = "$STORAGE_KEY"
EOF

    echo "✅ Backend infrastructure created"
}

create_backend

# Phase 5: Create Terraform Project
echo ""
echo "=== PHASE 5: CREATING TERRAFORM PROJECT ==="
create_terraform_project() {
    echo "Creating Terraform project structure..."

    # Create directory
    mkdir -p $PROJECT_NAME
    cd $PROJECT_NAME

    # Create backend.tf
    cat > backend.tf << EOF
terraform {
  backend "azurerm" {
    resource_group_name  = "$RG_NAME"
    storage_account_name = "$STORAGE_NAME"
    container_name       = "$CONTAINER_NAME"
    key                  = "terraform.tfstate"
  }
}
EOF
```

```
    # Create providers.tf
    cat > providers.tf << EOF
terraform {
  required_version = ">= 1.0.0"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
  }
}

provider "azurerm" {
  features {}
}
EOF

    # Create main.tf
    cat > main.tf << EOF
# Welcome to your Terraform project!
# This is a simple starter configuration.

resource "azurerm_resource_group" "example" {
  name     = "rg-\${var.environment}-\${var.project_name}"
  location = var.location

  tags = {
    Environment = var.environment
    ManagedBy   = "Terraform"
    Project     = var.project_name
  }
}

output "resource_group_name" {
```

```
  value = azurerm_resource_group.example.name
}
EOF

    # Create variables.tf
    cat > variables.tf << EOF
variable "environment" {
  type        = string
  description = "Environment name"
  default     = "dev"
}

variable "project_name" {
  type        = string
  description = "Project name"
  default     = "$PROJECT_NAME"
}

variable "location" {
  type        = string
  description = "Azure region"
  default     = "$REGION"
}
EOF

    # Create terraform.tfvars
    cat > terraform.tfvars << EOF
environment = "dev"
project_name = "$PROJECT_NAME"
location = "$REGION"
EOF

    # Create .gitignore
    cat > .gitignore << 'EOF'
# Local .terraform directories
**/.terraform/*
```

```
# .tfstate files
*.tfstate
*.tfstate.*

# Crash log files
crash.log

# Exclude all .tfvars files
*.tfvars
*.tfvars.json

# Ignore override files
*_override.tf
*_override.tf.json

# Include tfplan files to ignore
*.tfplan

# Ignore CLI configuration files
.terraformrc
terraform.rc
EOF

    echo "✅ Terraform project created"
}

create_terraform_project

# Phase 6: Initialize and Test
echo ""
echo "=== PHASE 6: INITIALIZING TERRAFORM ==="
initialize_terraform() {
    echo "Initializing Terraform..."

    # Export environment variables
```

```
    export ARM_CLIENT_ID
    export ARM_CLIENT_SECRET
    export ARM_TENANT_ID
    export ARM_SUBSCRIPTION_ID

    # Initialize
    terraform init

    # Validate
    terraform validate

    # Plan
    terraform plan -out=tfplan

    echo ""
    read -p "Apply the configuration? (yes/no): " APPLY_CONFIRM

    if [ "$APPLY_CONFIRM" = "yes" ]; then
        terraform apply tfplan
        echo "✅ Terraform configuration applied!"
    else
        echo "⚠️  Skipping apply. You can run 'terraform apply' later."
    fi

    rm -f tfplan
}

initialize_terraform

# Summary
echo ""
echo "================================================"
echo "        SETUP COMPLETE! 🎉"
echo "================================================"
echo ""
echo "✅ What was created:"
```

```
echo "   1. Azure Service Principal for Terraform"
echo "   2. Resource Group for state management"
echo "   3. Storage Account with container"
echo "   4. Terraform project structure"
echo ""
echo "📁 Project location: $PROJECT_NAME/"
echo ""
echo "🔐 Credentials saved to:"
echo "   - .env (Service Principal)"
echo "   - backend.info (Storage details)"
echo ""
echo "🚀 Next steps:"
echo "   1. Review the created resources in Azure Portal"
echo "   2. Add your infrastructure to main.tf"
echo "   3. Commit to version control:"
echo "      git init"
echo "      git add ."
echo "      git commit -m 'Initial Terraform configuration'"
echo "   4. Set up CI/CD (see .github/workflows/terraform.yml)"
echo ""
echo "⚠️  IMPORTANT SECURITY NOTES:"
echo "   - Keep .env and backend.info files SECURE"
echo "   - Do NOT commit them to version control"
echo "   - Consider using Azure Key Vault for production"
echo ""
echo "Need help? Check README.md for usage instructions."
```

7.2 Quick Start Commands

bash

```
# ONE COMMAND SETUP (if you trust the script)
curl -s https://raw.githubusercontent.com/your-repo/terraform-setup/main/setup.sh | bash

# OR download and run
```

```
wget https://raw.githubusercontent.com/your-repo/terraform-setup/main/setup.sh
chmod +x setup.sh
./setup.sh
```

---

8. Troubleshooting Common Issues

8.1 Authentication Issues

bash

```
# Problem: "Error: building AzureRM Client"
# Solution:

# 1. Check if logged in
az account show

# 2. Check environment variables
echo $ARM_CLIENT_ID
echo $ARM_CLIENT_SECRET
echo $ARM_TENANT_ID

# 3. Test service principal login
az login --service-principal \
  --username $ARM_CLIENT_ID \
  --password $ARM_CLIENT_SECRET \
  --tenant $ARM_TENANT_ID

# 4. Check role assignments
az role assignment list --assignee $ARM_CLIENT_ID

# 5. Reset credentials if needed
az ad sp credential reset --name $ARM_CLIENT_ID
```

8.2 Storage Account Issues

bash

```
# Problem: "Error: Failed to create storage container"
# Solution:

# 1. Check storage account exists
az storage account show --name $STORAGE_NAME

# 2. Check network rules (if using firewall)
az storage account network-rule list --account-name $STORAGE_NAME

# 3. Temporarily allow all networks (for testing)
az storage account update --name $STORAGE_NAME --default-action Allow

# 4. Check if container already exists
az storage container list --account-name $STORAGE_NAME

# 5. Check storage account key
az storage account keys list --account-name $STORAGE_NAME
```

8.3 Terraform Init Issues

bash

```
# Problem: "Error: Failed to get existing workspaces"
# Solution:

# 1. Delete local .terraform directory
rm -rf .terraform

# 2. Reinitialize with -reconfigure
terraform init -reconfigure

# 3. Check backend configuration
cat backend.tf
```

```
# 4. Test storage access
az storage blob list --account-name $STORAGE_NAME --container-name $C
ONTAINER_NAME

# 5. Check for state file corruption
az storage blob download --account-name $STORAGE_NAME \
    --container-name $CONTAINER_NAME \
    --name terraform.tfstate \
    --file terraform.tfstate.backup
```

8.4 Permission Issues

bash

```
# Problem: "Error: authorization failed"
# Solution:

# 1. Check Service Principal permissions
az role assignment list --assignee $ARM_CLIENT_ID

# 2. Add missing permissions
az role assignment create \
    --assignee $ARM_CLIENT_ID \
    --role "Contributor" \
    --scope "/subscriptions/$SUBSCRIPTION_ID"

# 3. Check if subscription is active
az account show --query "{State:state}"

# 4. Check for resource provider registration
az provider list --query "[].namespace"

# 5. Register missing providers
az provider register --namespace Microsoft.Storage
az provider register --namespace Microsoft.Network
```

9. Security Best Practices Checklist

✅ Authentication & Authorization:

- Use Service Principals, not user accounts for automation
- Implement least privilege (don't use Contributor for everything)
- Store secrets in Azure Key Vault
- Use Managed Identities where possible
- Rotate credentials regularly (every 90 days)

✅ State Management Security:

- Enable encryption at rest for storage account
- Enable soft delete for state files
- Enable versioning for state files
- Restrict network access to storage account
- Use private endpoints for production

✅ Code Security:

- Never commit secrets to version control
- Use .gitignore for sensitive files
- Implement pre-commit hooks
- Use Terraform security scanning (tfsec, checkov)
- Regular dependency updates

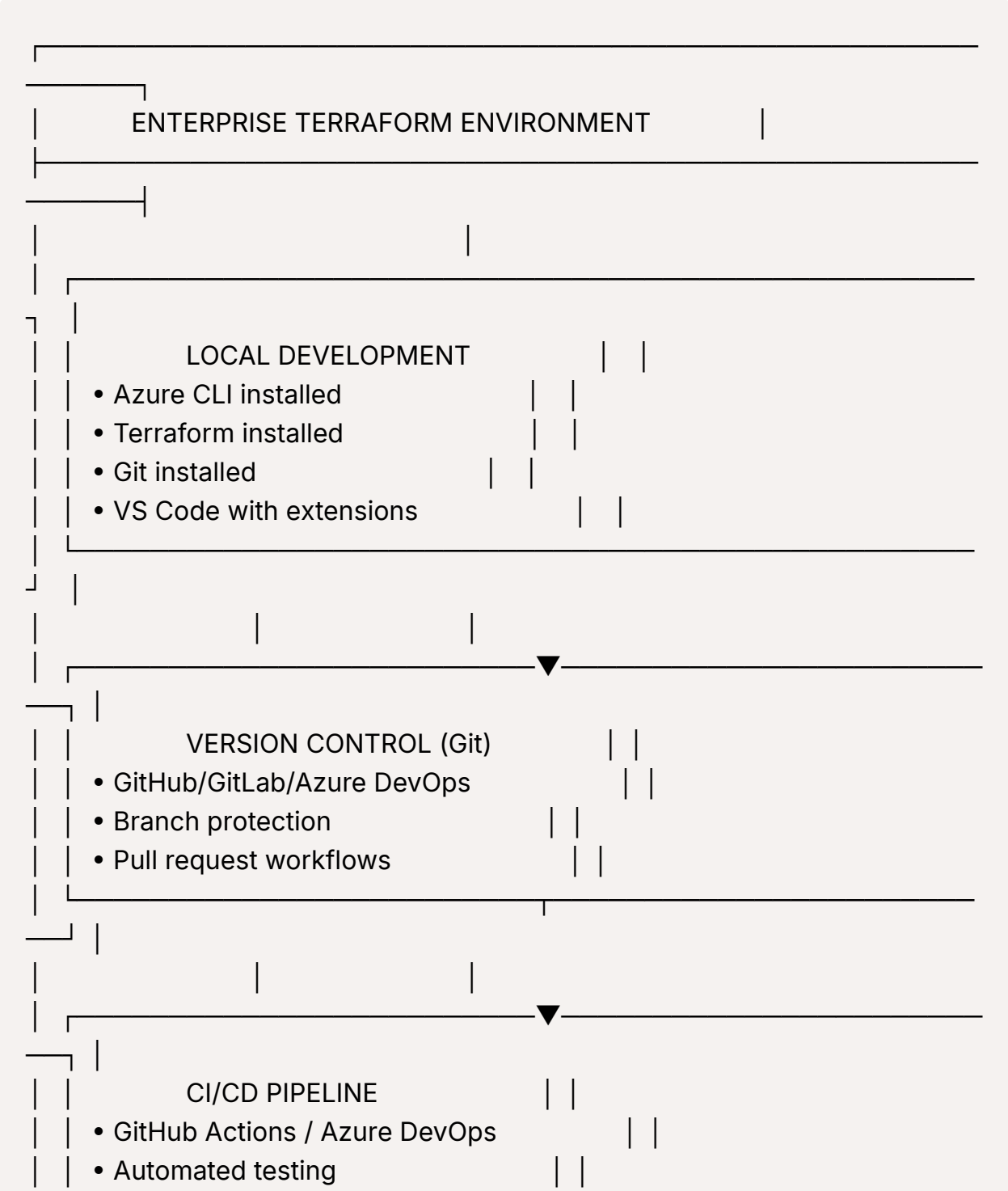
✅ Infrastructure Security:

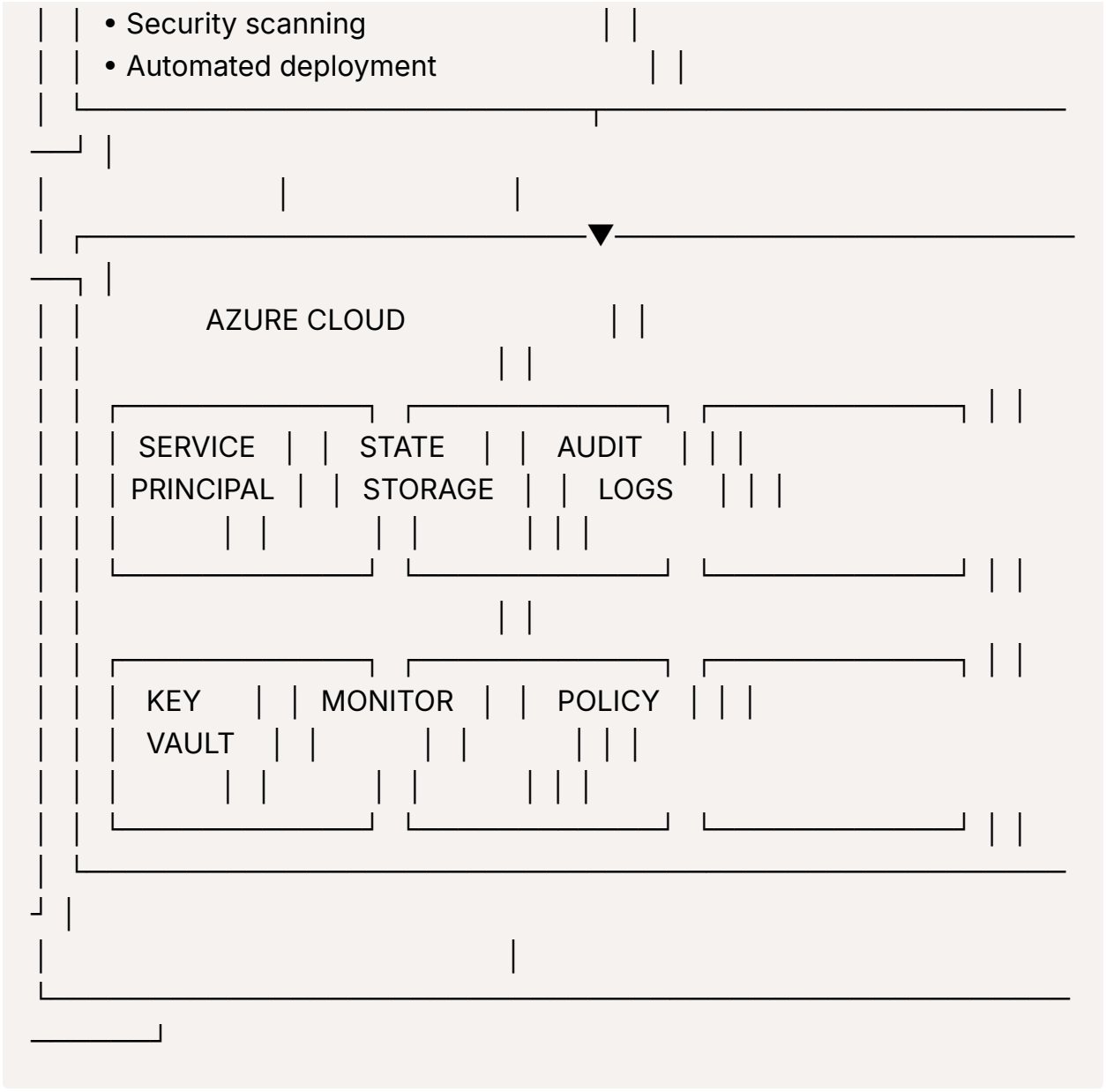
- Enable Azure Defender for Cloud
- Implement Azure Policy for compliance
- Enable logging and monitoring
- Regular security audits

- Disaster recovery testing

10. Complete Reference Architecture

text





Summary: Complete Setup Verification

✅ Final Verification Checklist:

bash

```
#!/bin/bash
# final-verification.sh
echo "=== FINAL SETUP VERIFICATION ==="
```

```
echo ""

echo "1. Tools Verification:"
echo "   Azure CLI: $(az --version 2>/dev/null | head -1 || echo 'Not found')"
echo "   Terraform: $(terraform --version 2>/dev/null | head -1 || echo 'Not fou
nd')"
echo "   Git: $(git --version 2>/dev/null || echo 'Not found')"
echo ""

echo "2. Azure Authentication:"
az account show --query "{User:user.name, Subscription:name, Tenant:tenan
tId}" 2>/dev/null || echo "   ❌ Not authenticated"
echo ""

echo "3. Service Principal:"
if [ -n "$ARM_CLIENT_ID" ]; then
    echo "   ✅ Service Principal configured"
    echo "   Client ID: ${ARM_CLIENT_ID:0:8}..."
else
    echo "   ❌ Service Principal not configured"
fi
echo ""

echo "4. Backend Resources:"
if az group show --name "terraform-state-*" &>/dev/null; then
    echo "   ✅ Resource Group exists"
else
    echo "   ❌ Resource Group not found"
fi
echo ""

echo "5. Terraform Project:"
if [ -f "main.tf" ] && [ -f "backend.tf" ]; then
    echo "   ✅ Terraform project structure exists"
else
    echo "   ❌ Terraform project incomplete"
```

```
fi
echo ""

echo "=== SETUP STATUS ==="
if command -v az &> /dev/null && \
   command -v terraform &> /dev/null && \
   [ -n "$ARM_CLIENT_ID" ] && \
   [ -f "main.tf" ]; then
    echo "✅ ENTERPRISE TERRAFORM SETUP COMPLETE!"
    echo ""
    echo "🚀 Ready to deploy infrastructure!"
else
    echo "❌ Setup incomplete. Please check above issues."
fi
```

Quick Reference Cheat Sheet

Essential Commands:

bash

```
# Installation
az login                                # Login to Azure
brew install azure-cli terraform          # Install tools (macOS)
apt install azure-cli terraform           # Install tools (Ubuntu)

# Service Principal
az ad sp create-for-rbac --name terraform-sp  # Create SP
az ad sp credential reset --name terraform-sp # Rotate credentials

# Backend Setup
az group create --name terraform-state-rg     # Create RG
az storage account create --name tfstateXXX   # Create storage
az storage container create --name tfstate    # Create container
```

```
# Terraform
terraform init                    # Initialize
terraform plan                    # Plan changes
terraform apply                   # Apply changes
terraform destroy                 # Destroy resources
terraform state list              # List resources in state

# Troubleshooting
terraform validate                # Validate configuration
terraform fmt                     # Format code
terraform providers               # Show providers
terraform graph | dot -Tpng > graph.png   # Generate dependency graph
```

Environment Variables:

bash

```
# Required for Service Principal authentication
export ARM_CLIENT_ID="00000000-0000-0000-0000-000000000000"
export ARM_CLIENT_SECRET="your-secret-here"
export ARM_TENANT_ID="00000000-0000-0000-0000-000000000000"
export ARM_SUBSCRIPTION_ID="00000000-0000-0000-0000-0000000000
00"

# Optional
export ARM_ENVIRONMENT="public"           # Azure environment
export ARM_SKIP_PROVIDER_REGISTRATION=true    # Skip auto-registration
export TF_LOG=DEBUG                       # Enable debug logging
```