Deploying with Terraform

Teacher Version

Software Architecture

March 6, 2023

Brae Webb

1 Before Class

Ensure you've had practice using the AWS Academy learner lab. It's preferable if you already have terraform installed[1]. Please also have one of IntelliJ IDEA, PyCharm, or VSCode with the terraform plugin installed. It is also helpful to have read the Infrastructure as Code lecture notes to understand the motivation for using a tool like Terraform.

2 This Week

This week we are going to put our Terraform and AWS knowledge together to deploy our Todo Application. Specifically, this week you need to:

- Authenticate Terraform to use the AWS learner lab.

- Configure an RDS database and use it in the Todo Application.

- Configure a single server website in Terraform and deploy the Todo Application to it.

3 Using Terraform in AWS Learner Labs

Following the steps from the week four practical, start a learner lab in AWS Academy. For this practical, you do not need to create any resources in the AWS Console. The console can be used to verify that Terraform has correctly provisioned resources.

1. Once the learner lab has started, click on 'AWS Details' to display information about the lab.

2. Click on the first 'Show' button next to 'AWS CLI' which will display a text block starting with `[default]`.

---

[1] https://learn.hashicorp.com/tutorials/terraform/install-cli

3. Create a directory for this week's practical.

4. Within that directory create a `credentials` file and copy the contents of the text block into the file. **Do not share this file contents — do not commit it**.

5. Create a `main.tf` file in the same directory with the following contents:

```
» cat main.tf
1  terraform {
2      required_providers {
3          aws = {
4              source = "hashicorp/aws"
5              version = "~> 3.0"
6          }
7      }
8  }

10  provider "aws" {
11      region = "us-east-1"
12      shared_credentials_file = "./credentials"
13  }
```

The `terraform` block specifies the required external dependencies, here we need to use the AWS provider. The `provider` block configures the AWS provider, instructing it which region to use and how to authenticate (using the credentials file we created).

6. We need to initialise terraform which will fetch the required dependencies. This is done with the `terraform init` command.

```
$ terraform init
```

This command will create a `.terraform` directory which stores providers and a provider lock file, `.terraform.lock.hcl`.

7. To verify that we have setup Terraform correctly, use `terraform plan`.

```
$ terraform plan
```

As we currently have no resources configured, it should find that no changes are required. Note that this does not ensure our credentials are correctly configured as Terraform has no reason to try authenticating yet.

4   Deploying a Database in AWS

Figure 1: Remote database deployment diagram

Now we have a locally running Todo App let's move to AWS, start up your Learner Lab environment now.

This is the last time we will heavily use the AWS user interface in the practicals. If you already feel confident in the AWS environment skip to Section **??** for the terraform setup.

To get started let's jump into the lab environment and have a look at AWS RDS which is an AWS managed database service. To get to the RDS service either search it or browse Services -> Database -> RDS as shown below.

Now we are in the management page for all our database instances, for today we just want to get a small instance running to explore the service. Head to "DB Instances (0/40)".

This page should appear familar as it's very similar to the AWS EC2 instance page. Let us create a new database by hitting the "Create Database" button.

We will be creating a standard database so select standard and MySQL. We will use version 8 to match the local version.

Choose a database creation method Info

○ Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

○ Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type **Info**

○ Amazon Aurora

● MySQL

○ MariaDB

○ PostgreSQL

○ Oracle

○ Microsoft SQL Server

Edition
● MySQL Community

ⓘ **Known issues/limitations**
Review the Known issues/limitations 🔗 to learn about potential compatibility issues with specific database versions.

Version

MySQL 8.0.27 ▼

For today we are going to use "Free Tier" but in the future, you may wish to explore the different deployment options. Please peruse the available different options.

For the teacher

Walk through what Multi-AZ means aka Multiple Availability Zones.

6

Templates

Choose a sample template to meet your use case.

○ **Production**
Use defaults for high availability and fast, consistent performance.

○ **Dev/Test**
This instance is intended for development use outside of a production environment.

● **Free tier**
Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.
Info

Availability and durability

Deployment options **Info**
The deployment options below are limited to those supported by the engine you selected above.

◉ Single DB instance (not supported for Multi-AZ DB cluster snapshot)
Creates a single DB instance with no standby DB instances.

◯ Multi-AZ DB instance (not supported for Multi-AZ DB cluster snapshot)
Creates a primary DB instance and a standby DB instance in a different AZ. Provides high availability and data redundancy, but the standby DB instance doesn't support connections for read workloads.

◯ Multi-AZ DB Cluster - new
Creates a DB cluster with a primary DB instance and two readable standby DB instances, with each DB instance in a different Availability Zone (AZ). Provides high availability, data redundancy and increases capacity to serve read workloads.

Now we need to name our database and create credentials to connect via. Please enter a reasonable password and keep this aside for later. We will need it for our local docker-compose file.

Settings

DB instance identifier Info
Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

```
todoapp-manual
```

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ **Credentials Settings**

Master username Info
Type a login ID for the master user of your DB instance.

```
todoapp
```

1 to 16 alphanumeric characters. First character must be a letter.

☐ **Auto generate a password**
  Amazon RDS can generate a password for you, or you can specify your own password.

Master password Info

```
••••••••••••••••••
```

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm password Info

```
••••••••••••••••••
```

We will use the default class type, t2.micro, which should be sufficient for this practical.

> **For the teacher**
>
> May want to mention that burstable is not recommended for consistantly used databases. Usually DBs are memory focused and thus the standard or memory optimised are used.

DB instance class

DB instance class Info

⦿ Standard classes (includes m classes)

⦿ Memory optimized classes (includes r and x classes)

🔵 Burstable classes (includes t classes)

```
db.t2.micro
1 vCPUs    1 GiB RAM    Not EBS Optimized        ▼
```

🔘 Include previous generation classes

For storage we will leave all the default options.

In connectivity we need to make sure our instance is publicly available. Usually you don't want to expose your databases publicly and, would instead, have a web server sitting in-front. But for today we will be running that web server locally so for convenience we need public access.

When selecting public access as yes we have to create a new Security Group, give this Security Group a sensible name.

Connectivity

Virtual private cloud (VPC) Info
VPC that defines the virtual networking environment for this DB instance.

Default VPC (vpc-07f8e8ea0408a9db9) ▼

Only VPCs with a corresponding DB subnet group are listed.

ⓘ After a database is created, you can't change its VPC.

Subnet group Info
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

default-vpc-07f8e8ea0408a9db9 ▼

Public access Info

🔘 Yes
Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.

◯ No
RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.

VPC security group
Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate incoming traffic.

◯ **Choose existing**
Choose existing VPC security groups

🔘 **Create new**
Create new VPC security group

New VPC security group name

todoapp-manual

Availability Zone Info

No preference ▼

▼ **Additional configuration**

Database port Info
TCP/IP port that the database will use for application connections.

3306

We will leave the authentication as password based but we need to expand the "Additional configuration". Fill in the "Initial Database Name" section to be "todoapp", this is similar to what we had in the Docker Compose.

Database authentication

Database authentication options Info

● Password authentication
  Authenticates using database passwords.

○ Password and IAM database authentication
  Authenticates using the database password and user credentials through AWS IAM users and roles.

○ Password and Kerberos authentication
  Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

▼ **Additional configuration**
  Database options, backup enabled, backtrack disabled, Enhanced Monitoring disabled, maintenance, CloudWatch Logs, delete protection disabled.

Database options

Initial database name Info

todoapp

If you do not specify a database name, Amazon RDS does not create a database.

DB parameter group Info

default.mysql8.0 ▼

Option group Info

default:mysql-8-0 ▼

Now we can click create which will take some time.

Estimated monthly costs

The Amazon RDS Free Tier is available to you for 12 months. Each calendar month, the free tier will allow you to use the Amazon RDS resources listed below for free:

- 750 hrs of Amazon RDS in a Single-AZ db.t2.micro Instance.
- 20 GB of General Purpose Storage (SSD).
- 20 GB for automated backup storage and any user-initiated DB Snapshots.

Learn more about AWS Free Tier. 🔗

When your free usage expires or if your application use exceeds the free usage tiers, you simply pay standard, pay-as-you-go service rates as described in the Amazon RDS Pricing page. 🔗

ⓘ You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

Cancel   **Create database**

Depending on your database it may take 10 to 30minutes to create, the larger and more complicated the setup the longer it usually takes. The database will also do a initial backup when its created.

RDS > Databases

Databases   🔵 Group resources   ↻   Modify   Actions ▾   Restore from S3   **Create database**

🔍 Filter by databases   ‹ 1 › ⚙

	DB identifier	▲	Role ▽	Engine ▽	Region & AZ ▽	Size ▽	Status ▽	CPU
○	todoapp-manual		Instance	MySQL Community	us-east-1f	db.t2.micro	🕐 Backing-up	▮▭▭▯1(

When the database has finished being created you can select it to view the configuration and details. In this menu we also see the endpoint address which we will need to copy into our docker compose file.

5   RDS Database with Terraform

Now would be a good time to browse the documentation for the RDS database in Terraform: `https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/db_instance`. Using our manual configuration, we can come up with a resource with the appropriate parameters as below:

```
» cat main.tf
1  locals {
2    password = "foobarbaz" # this is bad
3  }

5  resource "aws_db_instance" "todoapp-database" {
6    allocated_storage = 20
7    max_allocated_storage = 1000
8    engine = "mysql"
9    engine_version = "8.0.27"
10   instance_class = "db.t2.micro"
11   name = "todoapp"
12   username = "todoapp"
```

13

```
13    password = local.password
14    parameter_group_name = "default.mysql8.0"
15    skip_final_snapshot = true
16    vpc_security_group_ids = [aws_security_group.todoapp-database.id]
17    publicly_accessible = true

19    tags = {
20      Name = "todoapp-database"
21    }
22  }
```

Remember to create an appropriate security group as we did through the user interface.

```
      » cat main.tf
1   resource "aws_security_group" "todoapp-database" {
2     name = "todoapp-database"
3     description = "Allow inbound MySQL traffic"

5     ingress {
6       from_port = 3306
7       to_port = 3306
8       protocol = "tcp"
9       cidr_blocks = ["0.0.0.0/0"]
10    }

12    egress {
13      from_port = 0
14      to_port = 0
15      protocol = "-1"
16      cidr_blocks = ["0.0.0.0/0"]
17      ipv6_cidr_blocks = ["::/0"]
18    }

20    tags = {
21      Name = "todoapp-database"
22    }
23  }
```

> TODO: Add a section where we connect to the database using our locally running todo app.

6   Container on AWS

As we mentioned in the Infrastructure as Code notes [1], in this course we will use Docker to configure machines and Terraform to configure infrastructure. AWS has the ability to deploy Docker containers using a service known as Elastic Container Service (ECS). Unfortunately, the AWS Learner Labs provided by AWS do not support ECS.

To resolve this issue, we have created a Terraform module which allows us to deploy Docker images on EC2 instances and abstract over the underlying implementation. The documentation and source for this Terraform module is available on Github: https://github.com/CSSE6400/terraform/tree/main/container.

Using the documentation of the module, combined with the environment variables we know our back-end requires based on the `docker-compose.yml` file, we can develop a resource as below.

```
» cat main.tf
module "todoapp-backend" {
  source = "git::https://github.com/CSSE6400/terraform//container"

  image = "ghcr.io/csse6400/todo-app:combined-latest"
  instance_type = "t2.micro"
  environment = {
    APP_ENV="local"
    APP_KEY="base64:8PQEPYGlTm1t3aqWmlAw/ZPwCiIFvdXDBjk3mhsom/A="
    APP_DEBUG="true"
    LOG_LEVEL="debug"
    DB_CONNECTION="mysql"
    DB_HOST=aws_db_instance.todoapp-database.address
    DB_PORT="3306"
    DB_DATABASE="todoapp"
    DB_USERNAME="todoapp"
    DB_PASSWORD=local.password
  }
  ports = {
    "80" = "8000"
  }
  security_groups = [aws_security_group.todoapp-backend.name]

  tags = {
    Name = "todoapp-backend"
  }
}
```

Note that we are passing the address of our remote database into the container as an environment variable. This is a module which requires a source. In our case, the source will be the Github repository created earlier. Also notice that we map port 80 of the EC2 machine to port 8000 within the container, we should create a security group to make the instance accessible.

```
» cat main.tf
resource "aws_security_group" "todoapp-backend" {
  name = "todoapp-backend"
  description = "Todo App HTTP and SSH access"

  ingress {
    from_port = 80
```

```
7     to_port = 80
8     protocol = "tcp"
9     cidr_blocks = ["0.0.0.0/0"]
10   }

12   ingress {
13     from_port = 22
14     to_port = 22
15     protocol = "tcp"
16     cidr_blocks = ["0.0.0.0/0"]
17   }

19   egress {
20     from_port = 0
21     to_port = 0
22     protocol = "-1"
23     cidr_blocks = ["0.0.0.0/0"]
24   }
25 }
```

You will also want to create an output block to expose the address of the instance.

```
    » cat main.tf
1 output "url" {
2   value = module.todoapp-backend.public_dns
3 }
```

This should give you a `main.tf` file which fully deploys a todo application. If you haven't been applying as we go, try and apply the Terraform file now. If you have any issues, ask your tutor for guidance.

References

[1] B. Webb, "Infrastructure as code," March 2022. https://csse6400.uqcloud.net/handouts/iac.pdf.