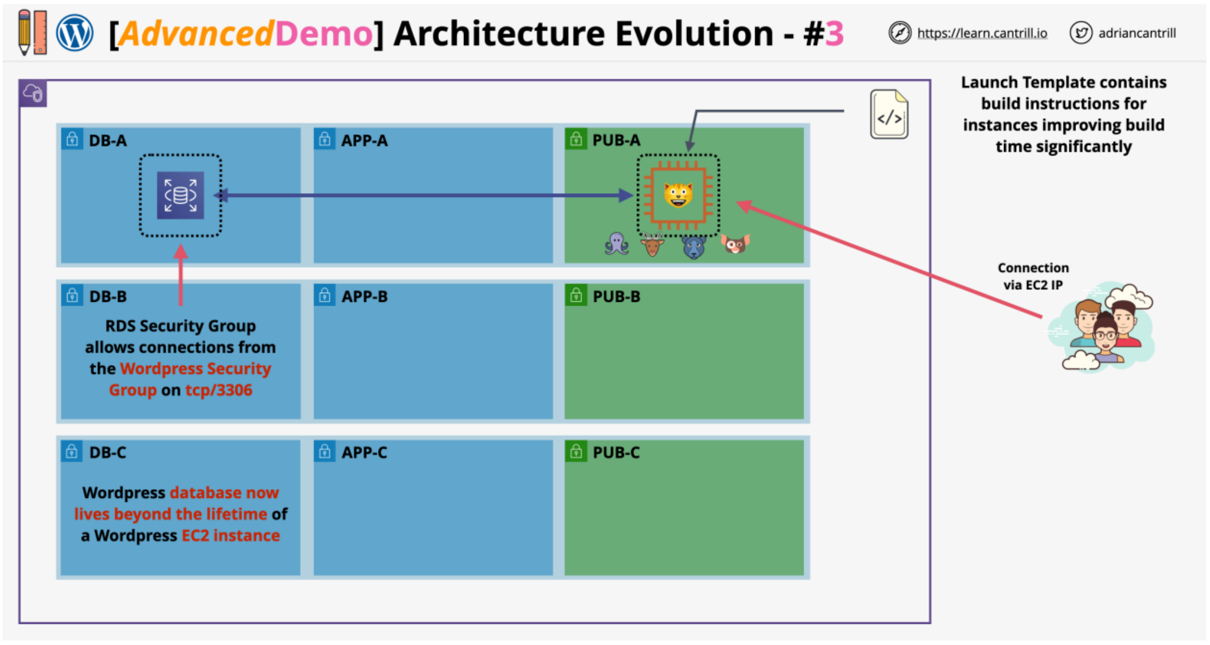
Advanced Demo - Web App - Single Server to Elastic Evolution


[AdvancedDemo] Architecture Evolution - #3
https://learn.cantrill.io
adriancantrill
DB-A
APP-A
PUB-A
DB-B
APP-B
PUB-B
DB-C
APP-C
PUB-C
RDS Security Group allows connections from the Wordpress Security Group on tcp/3306
Wordpress database now lives beyond the lifetime of a Wordpress EC2 instance
Launch Template contains build instructions for instances improving build time significantly
Connection via EC2 IP

In stage 3 you will be splitting out the database functionality from the EC2 instance .. running MariaDB to an RDS instance running the MySQL Engine.
This will allow the DB and Instance to scale independently, and will allow the data to be secure past the lifetime of the EC2 instance.

STAGE 3A - Create RDS Subnet Group

A subnet group is what allows RDS to select from a range of subnets to put its databases inside
In this case you will give it a selection of 3 subnets sn-db-A / B and C
RDS can then decide freely which to use.

Move to the RDS Console https://console.aws.amazon.com/rds/home?region=us-east-1#
Click Subnet Groups
Click Create DB Subnet Group
Under Name enter WordPressRDSSubNetGroup
Under Description enter RDS Subnet Group for WordPress
Under VPC select A4LVPC

Under Add subnets In Availability Zones select us-east-1a & us-east-1b & us-east-1c
Under Subnets check the box next to

- 10.16.16.0/20 (this is sn-db-A)
- 10.16.80.0/20 (this is sn-db-B)
- 10.16.144.0/20 (this is sn-db-C)

Click Create

STAGE 3B - Create RDS Instance

In this sub stage of the demo, you are going to provision an RDS instance using the subnet group to control placement within the VPC.

Normally you would use multi-az for production, to keep costs low, for now you should use a single AZ as per the instructions below.

Click Databases
Click Create Database
Click Standard Create
Click MySql
Under Version select MySQL 8.0.32

Scroll down and select Free Tier under templates *this ensures there will be no costs for the database but it will be single AZ only*

under Db instance identifier enter a4lWordPress under Master Username enter a4lwordpressuser under Master Password and Confirm Password enter 4n1m4l54L1f3

Under DB Instance size, then DB instance class, then Burstable classes (includes t classes) make sure db.t3.micro or db.t2.micro or db.t4g.micro is selected

Scroll down, under Connectivity, Compute resource select Don't connect to an EC2 compute resource
under Connectivity, Network type select IPv4
under Connectivity, Virtual private cloud (VPC) select A4LVPC
under Subnet group that wordpressrdssubnetgroup is selected
Make sure Public Access is set to No
Under VPC security groups make sure choose existing is selected, remove default and add A4LVPC-SGDatabase
Under Availability Zone set us-east-1a

IMPORTANT .. DON'T MISS THIS STEP Scroll down past Database Authentication & Monitoring and expand Additional configuration
in the Initial database name box enter a4lwordpressdb
Scroll to the bottom and click create Database

** this will take anywhere up to 30 minutes to create ... it will need to be fully ready before you move to the next step - coffee time !!!! **

STAGE 3C - Migrate WordPress data from MariaDB to RDS

Open the EC2 Console https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#Home:
Click Instances
Locate the WordPress-LT instance, right click, Connect and choose Session Manager and then click Connect
Type sudo bash
Type cd
Type clear

Populate Environment Variables

You're going to do an export of the SQL database running on the local ec2 instance

First run these commands to populate variables with the data from Parameter store, it avoids having to keep locating passwords

DBPassword=$(aws ssm get-parameters --region us-east-1 --names /A4L/Wordpress/DBPassword --with-decryption --query Parameters[0].Value)

```
DBPassword=`echo $DBPassword | sed -e 's/^"//' -e 's/"$//'`


DBRootPassword=$(aws ssm get-parameters --region us-east-1 --names /A4L/Wordpress/DBRootPassword --with-decryption --query Parameters[0].Value)

DBRootPassword=`echo $DBRootPassword | sed -e 's/^"//' -e 's/"$//'`


DBUser=$(aws ssm get-parameters --region us-east-1 --names /A4L/Wordpress/DBUser --query Parameters[0].Value)

DBUser=`echo $DBUser | sed -e 's/^"//' -e 's/"$//'`


DBName=$(aws ssm get-parameters --region us-east-1 --names /A4L/Wordpress/DBName --query Parameters[0].Value)

DBName=`echo $DBName | sed -e 's/^"//' -e 's/"$//'`


DBEndpoint=$(aws ssm get-parameters --region us-east-1 --names /A4L/Wordpress/DBEndpoint --query Parameters[0].Value)

DBEndpoint=`echo $DBEndpoint | sed -e 's/^"//' -e 's/"$//'`
```

Take a Backup of the local DB

To take a backup of the database run

```
mysqldump -h $DBEndpoint -u $DBUser -p$DBPassword $DBName > a4lWordPress.sql
```

** in production you wouldnt put the password in the CLI like this, its a security risk since a ps -aux can see it .. but security isnt the focus of this demo its the process of rearchitecting **

Restore that Backup into RDS

Move to the RDS Console https://console.aws.amazon.com/rds/home?region=us-east-1#databases:
Click the a4lWordPressdb instance
Copy the endpoint into your clipboard
Move to the Parameter store https://console.aws.amazon.com/systems-manager/parameters?region=us-east-1
Check the box next to /A4L/Wordpress/DBEndpoint and click Delete (please do delete this, not just edit the existing one)
Click Create Parameter

Under Name enter /A4L/Wordpress/DBEndpoint
Under Descripton enter WordPress DB Endpoint Name
Under Tier select Standard
Under Type select String
Under Data Type select text

Under Value enter the RDS endpoint endpoint you just copied
Click Create Parameter

Update the DbEndpoint environment variable with

DBEndpoint=$(aws ssm get-parameters --region us-east-1 --names /A4L/Wordpress/DBEndpoint --query Parameters[0].Value)

DBEndpoint=`echo $DBEndpoint | sed -e 's/^"//' -e 's/"$//'`

Restore the database export into RDS using

mysql -h $DBEndpoint -u $DBUser -p$DBPassword $DBName < a4lWordPress.sql

Change the WordPress config file to use RDS

this command will substitute localhost in the config file for the contents of $DBEndpoint which is the RDS instance

sudo sed -i "s/'localhost'/'$DBEndpoint'/g" /var/www/html/wp-config.php

STAGE 3D - Stop the MariaDB Service

sudo systemctl disable mariadb

sudo systemctl stop mariadb

STAGE 3E - Test WordPress

Move to the EC2 Console https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#Instances:sort=desc:tag:Name
Select the WordPress-LT Instance
copy the IPv4 Public IP into your clipboard
Open the IP in a new tab
You should see the blog, working, even though MariaDB on the EC2 instance is stopped and disabled
Its now running using RDS

STAGE 3F - Update the LT so it doesnt install

Move to the EC2 Console https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#Home:
Under Instances click Launch Templates
Select the WordPress launch Template (select, dont click) Click Actions and Modify Template (Create new version)
This template version will be based on the existing version ... so many of the values will be populated already
Under Template version description enter Single server App Only - RDS DB

Scroll down to Advanced details and expand it
Scroll down to User Data and expand the text box as much as possible

Locate and remove the following lines

systemctl enable mariadb

systemctl start mariadb

mysqladmin -u root password $DBRootPassword

```
echo "CREATE DATABASE $DBName;" >> /tmp/db.setup

echo "CREATE USER '$DBUser'@'localhost' IDENTIFIED BY '$DBPassword';" >> /tmp/db.setup

echo "GRANT ALL ON $DBName.* TO '$DBUser'@'localhost';" >> /tmp/db.setup

echo "FLUSH PRIVILEGES;" >> /tmp/db.setup

mysql -u root --password=$DBRootPassword < /tmp/db.setup

rm /tmp/db.setup
```

Click Create Template Version
Click View Launch Template
Select the template again (dont click) Click Actions and select Set Default Version
Under Template version select 2
Click Set as default version

STAGE 3 - FINISH

This configuration has several limitations :-

- ~~The application and database are built manually, taking time and not allowing automation~~ FIXED
- ~~^^ it was slow and annoying ... that was the intention.~~ FIXED
- ~~The database and application are on the same instance, neither can scale without the other~~ FIXED
- ~~The database of the application is on an instance, scaling IN/OUT risks this media~~ FIXED
- The application media and UI store is local to an instance, scaling IN/OUT risks this media
- Customer Connections are to an instance directly ... no health checks/auto healing
- The IP of the instance is hardcoded into the database ....

You can now move onto STAGE 4