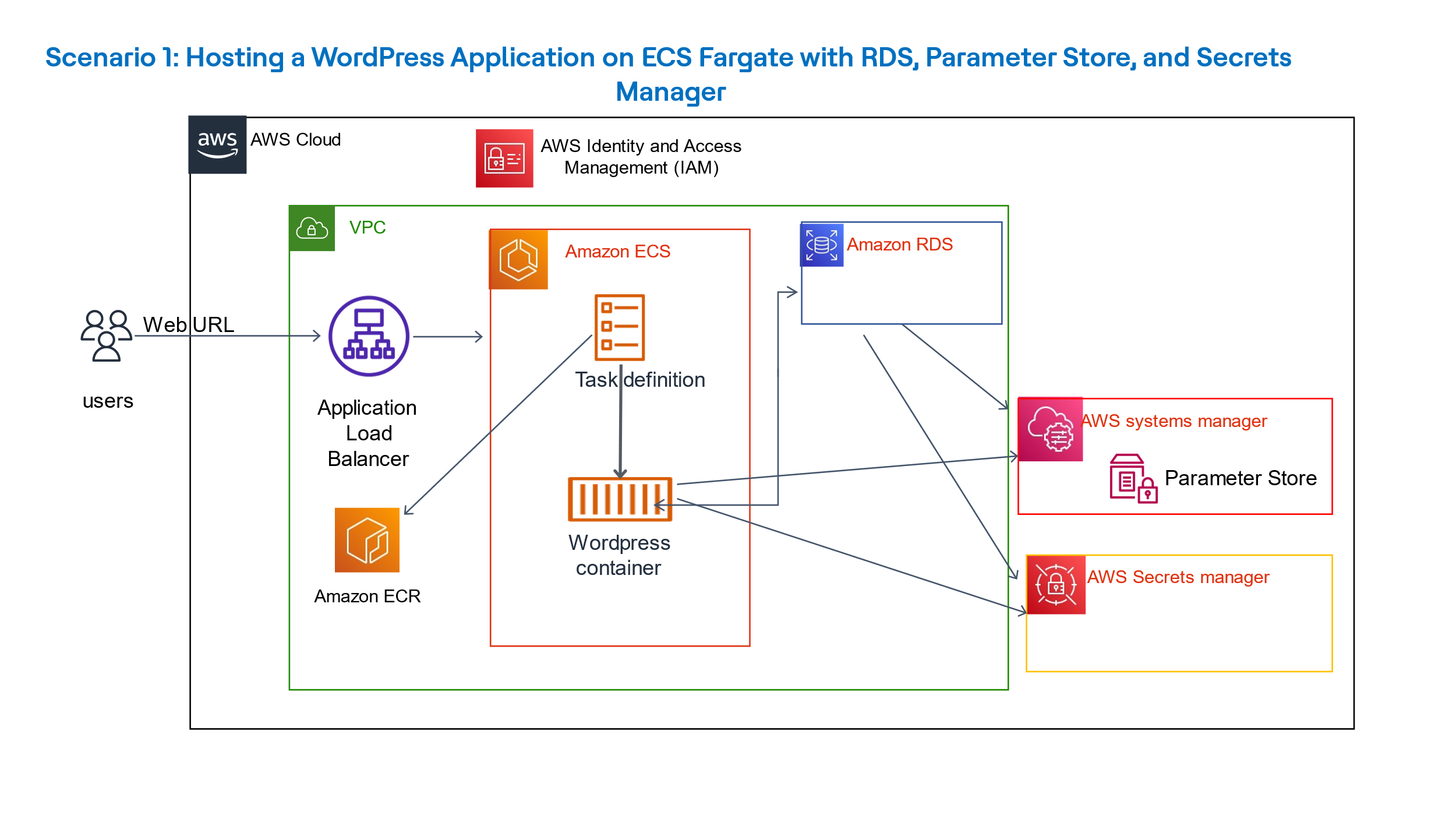
**Scenario 1: Hosting a WordPress Application on ECS Fargate with RDS, Parameter Store, and Secrets Manager**

**AWS services: ECS, ECR, Application load balancer, IAM, RDS, Secrets manager, parameter store, VPC, Subnets**

**Description:** You are working as an AWS DevOps engineer for abc private Ltd. and have been assigned to deploy application on ECS. You are required to follow best practices keeping in mind there is no comprise to security and compliance. Application should be accessed via ALB. Username and password should be stored in secret manager.



**Areas of scoring:**

1. Completion of tasks.
2. Quality of code and AWS recommendations implemented.
3. Security and compliance.
4. Troubleshooting and fixing ability.

**Tasks:**

1. Create two IAM roles with below details:

**Role 1: EcsTaskExecutionRole:**

Attach/Create below two policies.

* [AmazonECSTaskExecutionRolePolicy](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2Fservice-role%2FAmazonECSTaskExecutionRolePolicy) (AWS Managed)
* [AccessSecretsAndParameters](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/details/OurEcsTaskExecutionRole/editPolicy/AccessSecretsAndParameters?step=addPermissions) (create inline policy)

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"secretsmanager:GetSecretValue"

],

"Resource": [

"arn:aws:secretsmanager:us-east-1:<AWSaccountID>:secret:rds\*"

],

"Effect": "Allow"

},

{

"Action": [

"ssm:GetParameters"

],

"Resource": [

"arn:aws:ssm:us-east-1: <AWSaccountID>:parameter/dev/\*"

],

"Effect": "Allow"

}

]

}

Role 2:[EcsTaskRole](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/details/OurEcsTaskRole):

Create and attach below policy.

* [AccessSecretsAndParameters](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/details/OurEcsTaskExecutionRole/editPolicy/AccessSecretsAndParameters?step=addPermissions) (create inline policy)

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"secretsmanager:GetSecretValue"

],

"Resource": [

"arn:aws:secretsmanager:us-east-1: <AWSaccountID>:secret:rds\*"

],

"Effect": "Allow"

},

{

"Action": [

"ssm:GetParameters"

],

"Resource": [

"arn:aws:ssm:us-east-1: <AWSaccountID>:parameter/dev/\*"

],

"Effect": "Allow"

}

]

}

1. Review the terraform config files and create a s3 bucket and DynamoDB table to store terraform state file and enable state locking.
2. Deploy the terraform under “ecsScenario-master” folder to create VPCs,subnets,ALB,target group for ALB, DB subet grp and security groups.
3. ALB created through terraform. <Review and check details>
4. Database subnet group created through terraform. <Review and check details>
5. Create the Amazon RDS Instance

* Engine type – MySQL
* default Edition selected, which should be MySQL Community.
* Templates - Free Tier.
* DB cluster identifier - wordpress.
* Master username – admin
* Manage master credentials in AWS Secrets Manager.
* default encryption key for the Secrets Manager credentials.
* DB instance class, select db.t4g.micro.
* Storage type - General Purpose SSD (gp3) – 20 GB
* Connectivity section, select Don’t connect to an EC2 compute resource
* Place the database in the VPC titled “ECS-VPC”
* Choose the database subnet group from the DB subnet group dropdown- “main”
* Set Public access to No
* VPC security group (firewall) – “ECS-security-group”
* Within the Database options set the Initial database name to wordpress.
* Leave the rest of the options set to the defaults.

1. Create the Parameter Store Parameters and Verify Secrets Manager Secret

* Create the following 2 parameters in parameter store .

1. /dev/WORDPRESS\_DB\_HOST – String type and value as “YOUR\_RDS\_ENDPOINT:3306”
2. /dev/WORDPRESS\_DB\_NAME – Secure string type and value as “wordpress”
3. Create the ECR Repository and Image:

* Create repository.
* Visibility: Private
* Repository name: wordpress
* KMS encryption: Disabled

Image Creation:

* Since you need to run Docker and AWS CLI commands, use OurCloud9Environment . you can use access key and secret key shared with you or create new one.
* Login to ECR using AWS CLI
* Pull the latest Docker image for wordpress
* Push the image to ECR repo.

1. Create the Amazon ECS Task Definition

* Create new task definition.
* Task definition family: wordpress-td
* Launch type: AWS Fargate
* Attach the Task role and Task Execution role.
* enter the following settings for Container details:
* Image: use the image uploaded in ECR

|  |  |  |
| --- | --- | --- |
| KEY | VALUE TYPE | VALUE |
| WORDPRESS\_DB\_HOST | ValueFrom | ARN of the respective Parameter Store parameter |
| WORDPRESS\_DB\_NAME | ValueFrom | ARN of the respective Parameter Store parameter |
| WORDPRESS\_DB\_USER | ValueFrom | ARN of the respective Secrets Manager RDS secret, specifying the specific key value by adding :username:: |
| WORDPRESS\_DB\_PASSWORD | ValueFrom | ARN of the respective Secrets Manager RDS secret, specifying the specific key value by adding :password:: |

1. Create the ECS Cluster and Service

* Create Cluster:
* Cluster name: Wordpress-Cluster
* Infrastructure: AWS Fargate (serverless)
* Create Service in your newly created Cluster:
* Compute options: Launch type
* Launch type: FARGATE.
* Platform: LATEST.
* Application type : Service.
* Family, under Task definition: wordpress-td with latest version
* Service name: wordpress-service
* desired tasks: 1
* Networking section: Your Custom VPC
* security group: ECS Security group
* Load balancer type: ALB and existing ALB
* Health check grace period: 30 seconds
* Listener values: Existing
* Target group: Existing

1. Validate whether the service is running and successfully registered under ALB. Use ALB DNS to access the site.

2) Write python code to get specific secret from secrets manager that match a specified name. once secret name is successfully fetched, split the name, and store the alphabets in array and count the occurrence of specific alphabet in that array list.

**Example:** secret is stored as “name:Database”.

Array should be [‘D’, ‘a’, ’t’, ’a’, ’b’, ’a’, ’s’, ’e’]

Occurrence of letter a = 3.

3) EKS Cluster: Launching an Amazon EKS Cluster

* IAM user with admin user is already created- k8-admin.
* Use EC2 server named , Linux jump-host which has pre-installed software such as AWS CLI, kubectl, git, eksctl.
* EKS cluster is created with name as “dev” and managed node group.

**Task:** a) create a service to access the pods.

b) create a deployment using nginx image .

Number of worker nodes should be 2 and nginx should be accessible via URL

4) Ansible:

* Install Nginx using Ansible Playbook.​
* Create an Amazon Linux 2 instance with t2.micro instance type.
* Install Ansible into this server. This server will work as Ansible controller, Provide ‘Ansible controller’ as Name for this server.

* Now create another server with t2.micro and name this server’ Web server’.
* Now create an Ansible playbook in Ansible controller server to install Nginx on ‘Web server’ and a sample index.html page with ‘Hello word’ text.

* Execute this playbook to install Nginx and index.html page in ‘Web server’.
* Now validate the web page via Ip address of server.