SETTING UP A KUBERNETES CLUSTER IN AWS USING TERRAFORM

Initialization:

This documentation offers a detailed tutorial for using Terraform to set up a Kubernetes cluster in AWS.

Requirements:

- An account with AWS.
- Terraform must be set up on your local computer.
- A basic understanding of Kubernetes and AWS services.

Procedures:

- **Configure AWS Login Information:**

- Make sure your local computer is set up with AWS access and secret keys or create an IAM role.

- **Making Terraform Configuration Files:**

- Your Terraform project should have a directory created.

- Create `.tf` files for your resources, such as VPC, subnets, security groups, etc., inside the directory.

- **Initialize Terraform:**

- To initialise Terraform and download necessary providers, run `terraform init` in your project directory.

- **Deploy Networking Resources:**

- Use Terraform configuration files to define VPC, subnets, and security groups.

- **Create the Kubernetes worker nodes:**

- Using Terraform, setup the worker node instances by defining the instance type, AMI, and networking information.

- **Generate a SSH Key Pair:**

- Generate the SSH key pair needed to connect to the worker nodes.

- **Set up the Kubernetes Control Plane:**

- To configure the Kubernetes control plane on a selected node, use kubeadm or equivalent tool.

- **Connect Worker Nodes to the Cluster:**

- Connect the worker nodes to the Kubernetes cluster using the token created during control plane setup.

- **Install kubectl:**

- To manage the Kubernetes cluster locally, install kubectl.

- **Set up kubectl:**

- Set up the Kubernetes cluster with kubectl.

- **Test the Cluster:**

- To ensure that every node is prepared and a part of the cluster, run `kubectl get nodes`.

- **Deploy Applications:**

- To deploy your applications to the cluster, use `kubectl`.

Conclusion:

You have just used Terraform to successfully set up a Kubernetes cluster in AWS. Your apps can now be managed and scaled effectively on this cluster.

Additional Considerations made during implementation:

- Configure an ingress controller to direct traffic from outside sources.

- Integrate appropriate security mechanisms, such as Kubernetes RBAC and IAM roles for EC2 instances.

- Explore tools like Helm for managing Kubernetes applications.

- Regularly update and maintain your cluster to ensure security and stability.

SETTING UP AN ELASTIC CONTAINER REGISTRY(ECR) IN AWS USING TERRAFORM

Initialization:

This documentation offers a step-by-step tutorial for configuring a Terraform-based Elastic Container Registry (ECR) in AWS.

Requirements:

- Terraform deployed locally with an AWS account.
- Knowledge of the fundamentals of containerization and AWS services.

Procedures:

- **Create AWS Login Credentials:**

- Make sure your local computer is set up with AWS access and secret keys or create an IAM role.

- **Making Terraform Configuration Files:**

- Your Terraform project should have a directory created.

- Create `.tf` files for your ECR and relevant materials inside the directory.

- **Initialize Terraform:**

- To initialise Terraform and download necessary providers, run `terraform init` in your project directory.

- **Setup ECR Repository:**

- Include the repository name and any optional settings when defining an ECR repository in your Terraform setup file.

- **Setup Repository Policy:**

- To manage access to your ECR repository, define a repository policy using Terraform. Access can be limited depending on IAM roles or AWS accounts.

- **Create a policy for the ECR lifecycle (This is optional):**

- To control the preservation and cleanup of images, define an ECR lifecycle policy in Terraform, if you like.

- **Generate an ECR Authentication Token:**

- Generate an authentication token using the AWS Command Line Interface (CLI) and safely authenticate Docker with your ECR repository.

- **Build and Push Docker Image:**

- Build your Docker image and add the URI of the ECR repository to it.

- To access ECR, log in using the authentication token generated in the previous step.

- Push the Docker image to your ECR repository.

- **Access ECR Image in Kubernetes (This is Optional):**

- Configure the Kubernetes deployment to use the ECR image if you're using it. This might entail generating an authentication Kubernetes secret.

Conclusion:

You have just used Terraform to successfully setup a Container Registry (ECR) as a private docker registry in AWS. Now, you can manage and securely store your Docker container images.

Additional Considerations made during implementation:

- To find vulnerabilities in your container images, set up image scanning.

- Apply IAM roles and policies to implement fine-grained access control.

- Consider using CI/CD pipelines to automate the build and deployment of images.

- Review and manage your ECR repositories frequently to maintain security and maximise storage efficiency.

SETTING UP A MySQL DATABASE IN AWS USING TERRAFORM

Initialization:

This documentation offers a step-by-step tutorial for configuring a MySQL database in AWS.

Requirements:

- AWS account.
- Install terraform on your local machine.

Procedures:

- **Create AWS Login Credentials:**

- Make sure your local computer is set up with AWS access and secret keys, or create an IAM role.

- **Making Terraform Configuration Files:**

- Your Terraform project should have a directory created.

- Create `.tf` files for your resources, such as Virtual Private Clouds (VPC), subnets, security groups, etc., inside the directory.

- **Initialize Terraform:**

- To initialise Terraform and download necessary providers, run `terraform init` in your project directory.

- **Deploy Networking Resources:**

- Use Terraform configuration files to define VPC, subnets, and security groups.

- **Set Up a Relational Database Service (RDS) Instance:**

- Using Terraform, create a MySQL RDS (Relational Database Service) instance.

- Indicate instance settings such as the database name, instance class, allocated storage, and engine version.

- **Setup Database Security:**

- To manage incoming and outgoing traffic to the RDS instance, define security group rules.

- If necessary, configure parameter groups to modify MySQL settings.

- **Create a password for the database:**

- Create a strong password for the MySQL database using Terraform.

- **Start the MySQL Database Instance:**

- To setup the RDS instance with the desired parameters and password, apply the Terraform configuration.

- **Connect the Database:**

- With the given endpoint, database name, and created password, connect to the RDS instance using a suitable MySQL client.

- **Setup the initial database:**

    - Create the tables, indexes, and other database objects that your application requires.

Conclusion:

You have just used Terraform to successfully set up a MySQL database in AWS. Now you can manage and store the data for your application using this database.

Additional Considerations made during implementation:

- Put good backup and recovery procedures in place for your MySQL database.

- To ensure the wellbeing and efficiency of the database instance, enable monitoring and warnings.

- Apply security updates and perform routine upkeep on your RDS instance to enhance security and performance.

- For high availability, consider enabling multi-AZ deployment and enabling automatic backups.