# Assignment 2: Design a 12 Factor App for an Ecommerce Website using AWS Tech Stack

# Background:

Your team is building an ecommerce website that needs to handle high traffic, be highly available, and scale easily. The app will be hosted on the AWS cloud, and you are tasked with designing a 12 Factor App architecture that follows best practices for scalability, reliability, and maintainability.

Instructions:

1. Design the application as a set of loosely-coupled services that communicate through APIs. Use AWS Lambda functions to run your application code, and deploy them using AWS Serverless Application Model (SAM).
2. Store your application's configuration in environment variables, and use AWS Parameter Store to securely store and manage sensitive data such as database passwords and API keys.
3. Use AWS Elastic Beanstalk for deployment and scaling of the web tier. Elastic Beanstalk provides an easy-to-use platform for deploying and scaling web applications, and can automatically handle load balancing and auto-scaling based on demand.
4. Use AWS RDS for your database tier. RDS provides a fully managed database service that is scalable, reliable, and secure. You can easily set up a multi-AZ deployment for high availability, and use read replicas for scaling read-heavy workloads.
5. Use AWS S3 for storing static assets such as images and videos. S3 provides a scalable, reliable, and cost-effective storage service that can handle unlimited amounts of data.
6. Use AWS CloudFront for content delivery and caching. CloudFront provides a global content delivery network that can accelerate the delivery of your website's static assets, and can cache content at the edge for faster access.
7. Use AWS API Gateway for creating and managing APIs. API Gateway provides a fully managed service that makes it easy to create, publish, and secure APIs at scale.
8. Use AWS CloudWatch for monitoring and logging. CloudWatch provides a central location for monitoring and logging your application's performance and health, and can trigger alerts and notifications based on custom metrics and thresholds.
9. Use AWS CodePipeline for continuous integration and deployment. CodePipeline provides a fully managed service for automating the build, test, and deployment of your application code.
10. Use AWS Identity and Access Management (IAM) for security and access control. IAM provides a secure and scalable way to manage access to your AWS resources, and can be used to set granular permissions and roles for users, groups, and services.
11. Use AWS KMS for encryption and key management. KMS provides a fully managed service for creating and managing encryption keys, and can be used to encrypt and decrypt data at rest and in transit.
12. Use AWS CloudFormation for infrastructure as code. CloudFormation provides a way to define and deploy infrastructure resources using code, and can be used to automate the provisioning and deployment of your application stack.

## Deliverables:

1. A high-level architecture diagram that shows the various AWS services used in your 12 Factor App, and how they are connected.
2. A detailed description of each of the 12 factors, and how they are implemented in your app.
3. A sample code snippet that shows how to use environment variables and AWS Parameter Store to securely store and manage configuration data.
4. A sample AWS SAM template that shows how to define and deploy a Lambda function and API Gateway endpoint.
5. A sample AWS CloudFormation template that shows how to define and deploy the entire application stack, including web tier, database tier, and other infrastructure resources.

# Solution:

1. Codebase: Use a single codebase for the application, version control, and build system. Store the code in a Git repository on AWS CodeCommit.
2. Dependencies: Explicitly declare and isolate dependencies using a package manager like npm or pip. Use AWS Elastic Container Registry (ECR) to store and manage container images for dependencies.
3. Config: Store configuration in environment variables, not in the code. Use AWS Systems Manager Parameter Store or AWS Secrets Manager to store secrets and configuration data.
4. Backing services: Treat backing services as attached resources that can be replaced or upgraded without impacting the application. Use AWS managed services like Amazon RDS for databases and Amazon S3 for file storage.
5. Build, release, run: Separate the build, release, and run stages of the application. Use AWS CodeBuild to build the application and AWS CodePipeline to automate the build, test, and deployment process.
6. Processes: Run the application as stateless processes that can be easily scaled horizontally. Use AWS Elastic Container Service (ECS) or AWS Fargate to manage containerized processes.
7. Port binding: Export services via a port binding mechanism. Use Elastic Load Balancing to distribute traffic across containers or instances.
8. Concurrency: Scale the application horizontally by adding more instances or containers. Use AWS Auto Scaling to automatically adjust capacity based on demand.
9. Disposability: Maximize robustness by minimizing the time it takes to start and stop the application. Use AWS Elastic Container Service (ECS) or AWS Fargate to manage containerized processes.
10. Dev/prod parity: Keep development, staging, and production environments as similar as possible. Use AWS CloudFormation or AWS Elastic Beanstalk to create and manage identical environments.
11. Logs: Treat logs as event streams and aggregate them in a centralized location. Use Amazon CloudWatch Logs to collect and analyze logs.
12. Admin processes: Run admin/management tasks as one-off processes. Use AWS Lambda to run serverless tasks on an event-driven basis.

Overall, the AWS tech stack provides many services that align well with the 12 Factor App principles, making it a good choice for building scalable and maintainable ecommerce applications.