AWS Infrastructure

Content

- Introduction to Cloud Computing
- Introduction to AWS
- Identity & Access
- EC2 & Lambda Compute Services
- AWS Elastic Load Balancing
- S3 Storage
- RDS & DynamoDB For Databases
- Networking & Domains
- CloudWatch Monitoring & Logging
- Devops & CI/CD Pipeline in AWS

Introduction To Cloud Computing

Cloud Computing

- **What Is Cloud Computing?**
  - Cloud computing refers to the delivery of computing services, such as servers, storage, databases, networking, software, analytics, and intelligence over the internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale.
  - cloud computing eliminates the need for you to own or manage physical servers, enabling you to build scalable and flexible applications using shared infrastructure provided by cloud vendors.

Cloud Computing

- **Key Characteristics:**
  - On-demand self-service: Instantly provision resources without human interaction.
  - Broad network access: Access cloud services over the internet from anywhere.
  - Rapid elasticity: Automatically scale resources up or down based on demand.
  - Measured service: Pay only for what you use.

Cloud Computing

- **Benefits of Cloud for Backend Developers**
  - Cloud computing brings major advantages for backend development by simplifying infrastructure management, reducing costs, and enabling global reach.
  - Cloud platforms allow backend developers to focus on building features, not managing infrastructure. You can release faster, serve more users, and stay secure by default.

Cloud Computing

1. **Scalability:**

   - Automatically scale servers and databases as traffic grows.
   - Avoid performance issues during peak usage.
   - Use services like Auto Scaling Groups (EC2) or Lambda for elastic scaling.

2. **Cost Efficiency:**

   - Pay only for what you use, no idle server costs.
   - Pricing models: on-demand, reserved, spot instances.
   - Eliminate upfront investment in hardware.

Cloud Computing

3. **Faster Development & Deployment:**

   - Provision new servers, databases, and load balancers in minutes.
   - Integrate CI/CD tools for automated deployments.
   - Quickly iterate and test without infrastructure setup delays.

4. **Global Reach:**

   - Deploy applications in multiple AWS regions worldwide.
   - Provide low-latency experience for users across the globe.
   - Use AWS Edge locations for caching and acceleration (CloudFront).

Cloud Computing

5. **Security & Compliance:**

   - Built-in security tools: IAM, KMS, VPC, Security Groups.
   - End-to-end encryption and access control.
   - Compliance with standards like ISO, SOC, GDPR, HIPAA (depending on region and setup).

Cloud Computing

- **Cloud Service Models:**
  - Cloud computing is divided into three main service models, each offering a different level of control and responsibility. As a backend developer, understanding these models helps you choose the right tool for each task.
  - **Service Models:**
    - **IaaS** - Infrastructure as a Service
    - **Paas** - Platform as a Service
    - **Saas** - Software as a Service

Cloud Computing

1. **IaaS – Infrastructure as a Service:**
    - Provides virtualized computing resources over the internet.
    - You manage: operating system, runtime, application code.
    - Cloud provider manages: hardware, networking, virtualization.
    - Example: Amazon EC2, Azure VMs, Google Compute Engine.
    - Use case: Full control over your stack, ideal for custom deployments.

Cloud Computing

2. **PaaS – Platform as a Service:**
    - Provides a managed platform to deploy and scale applications.
    - You manage: application code and data.
    - Cloud provider manages: runtime, OS, networking, scaling.
    - Example: AWS Elastic Beanstalk, Heroku, Google App Engine.
    - Use case: Deploy your backend app without managing servers.

Cloud Computing

3. **SaaS – Software as a Service:**

   - Delivers software over the internet; no need to install or manage infrastructure.
   - You manage: usage only.
   - Cloud provider manages everything else.
   - Example: GitHub, Google Docs, Firebase Auth, SendGrid.
   - Use case: Use third-party tools (e.g. Auth, storage, payments) without building them.

Cloud Computing

Choosing the right model

Model	You Manage	Examples	Control Level
IaaS	OS + Code	EC2, DigitalOcean	High
PaaS	Just Code	Elastic Beanstalk	Medium
SaaS	Nothing	Firebase Auth	Low

Introduction To AWS

Amazon Web Services

- **What Is AWS?**
  - AWS (Amazon Web Services) is the world's most comprehensive and widely adopted cloud platform, offering over 200 fully featured services from data centers globally.
  - AWS is not just a hosting platform, it's an entire ecosystem that allows you to build, scale, and secure backend applications at any size.
  - Offers compute, storage, databases, networking, analytics, ML, IoT, DevOps tools, and more
  - Powering major companies like Netflix, NASA, Twitch, Airbnb, and Spotify

Amazon Web Services

- Why Developers Choose AWS:
  - Huge service ecosystem: everything from virtual machines to machine learning
  - Reliable and secure: 99.99%+ uptime, global compliance standards
  - Flexible pricing: free tier, pay-as-you-go, reserved and spot instances
  - Seamless integrations with CI/CD, container tools, and SDKs

Amazon Web Services

- AWS Global Infrastructure:
  - AWS is built to support high availability, low latency, and global scale through a vast network of data centers.
  - **Regions:**
    - A Region is a physical location in the world with multiple data centers.
    - Examples: us-east-1 (N. Virginia), eu-central-1 (Frankfurt), af-south-1 (Cape Town)
    - Each region is isolated to improve fault tolerance and compliance

Amazon Web Services

- **Availability Zones (AZs):**
  - A Region consists of 2–6 AZs.
  - An AZ is one or more data centers with independent power, networking, and cooling.
  - Services like RDS, EC2, and Elastic Load Balancers can span AZs for high availability
- **Edge Locations:**
  - Edge locations are content delivery endpoints used by CloudFront (AWS CDN)
  - Used to cache and deliver content closer to users for faster response times

Amazon Web Services

- **AWS Free Tier Overview:**
  - The AWS Free Tier allows new users to explore AWS services at no cost, making it ideal for students, developers, and startups.
  - There are mainly three types of free tier:
    1. 12 month free tier
    2. Always free
    3. Short term trials

Amazon Web Services

- **12-Month Free Tier:**
  - Valid for 12 months after account creation
  - once you have exhausted your Free Tier Credits, whichever comes first
- **Examples:**
  - EC2: 750 hours/month of t2.micro or t3.micro (across all instances)
  - RDS: 750 hours/month of db.t2.micro + 20 GB storage
  - S3: 5 GB standard storage + 20K GET and 2K PUT requests

Amazon Web Services

- **Always Free:**
  - No expiration (as long as you're within limits)
- **Examples:**
  - Lambda: 1M free requests/month, 400,000 GB-seconds compute
  - CloudWatch: 10 custom metrics, 5GB log ingestion/month
  - DynamoDB: 25 GB storage + 200M requests/month

Amazon Web Services

- **Short-Term Trials:**
  - Try premium services for a limited time (e.g., AWS SageMaker Studio for 2 months)
- **How to Stay Within Free Tier:**
  - Use Billing Dashboard and Budgets
  - Set alerts to track usage and costs
  - Avoid launching services outside the free tier (e.g. larger EC2 instances)

Amazon Web Services

- **Backend Use Cases with AWS:**
  - **REST API Hosting:**
    - Use EC2 or Elastic Beanstalk to host Node.js/Express or Django apps
    - Use API Gateway + Lambda for serverless endpoints
    - Load balancers and autoscaling for high-traffic APIs
  - **File Storage & Media Uploads:**
    - Use S3 to store images, videos, PDFs, and static assets
    - Generate pre-signed URLs for secure, temporary uploads/downloads
    - Configure S3 event triggers for Lambda-based processing (e.g., image resizing)

Amazon Web Services

- **Authentication & User Management:**
  - Use Cognito for signup/login/OAuth2
  - Integrate with IAM for access control in admin dashboards
- **Databases & Caching:**
  - Use RDS for managed SQL (PostgreSQL/MySQL)
  - Use DynamoDB for NoSQL workloads
  - Add ElastiCache (Redis/Memcached) for performance

Amazon Web Services

- **Logging, Monitoring, & Alerts:**
  - Log app and system events to CloudWatch Logs
  - Create dashboards to monitor latency, memory, error rates
  - Setup CloudWatch Alarms to get notified via SNS
- **CI/CD Pipelines:**
  - Build and deploy backend apps using CodePipeline, CodeBuild, and GitHub
  - Automate testing, building, and deployment to EC2 or Lambda

Identity & Access

Identity & Access

- **What Is IAM? (Identity and Access Management)**
    - IAM is AWS's permission management system. It controls who can access what, and what actions they can perform on AWS services.
- **Core Functions:**
    - Authentication: Verifying who the user is
    - Authorization: Defining what the user can do
    - IAM lets you securely manage access to:
        - EC2, S3, RDS, Lambda, etc.
        - Billing, resource management, logs

Identity & Access

- **IAM Users, Groups, Policies, and Roles:**
  - **IAM User:**
    - A person or app needing access to AWS (e.g., dev-user, ci-bot)
    - Has long-term credentials (access keys/password)
  - **IAM Group:**
    - A collection of users with shared permissions
    - Example: Developers, Admins
  - **IAM Policy:**
    - A JSON document that defines permissions
    - Example: Allow s3:GetObject on a bucket

Identity & Access

IAM Users VS IAM Roles

Feature	IAM User	IAM Role
Identity Type	Human or programmatic	Temporary assumed identity
Credentials	Long-term (Access keys)	Temporary (Auto-rotated)
Use Case	Admin, developers, CI/CD tools	EC2, Lambda, ECS tasks
Trust Mechanism	Authenticated via login or API keys	Assumed by trusted entities (e.g., EC2)

Identity & Access

- Best Practices for IAM in Backend Apps:
  - Least Privilege Principle
    - Only give the exact permissions needed, nothing more
  - Use IAM Roles for EC2, Lambda, ECS
    - Don’t hardcode AWS credentials, attach a role instead
  - Rotate Access Keys Regularly (if used)

Identity & Access

- Best Practices for IAM in Backend Apps:
  - Enable MFA on Root and Admin Users
    - Adds extra security for critical accounts
  - Audit with IAM Access Analyzer + CloudTrail
    - See who accessed what, when, and from where
  - Tag IAM Users and Roles
    - Add metadata like Environment=dev, Team=backend to help track usage

Compute Services

EC2 Compute Services

- **What Is EC2 (Elastic Compute Cloud)?**
  - EC2 is AWS's virtual server service. It lets you launch and manage scalable virtual machines (instances) in the cloud.
  - **Key Concepts:**
  - Instance: Virtual machine (like Ubuntu or Amazon Linux)
  - Choose CPU, memory, storage (instance types like t3.micro, m5.large)
  - Fully customizable: install Node.js, databases, etc.

EC2 Compute Services

- **Use Cases:**
  - Hosting backend apps, APIs
  - Running databases, cron jobs
  - Custom server environments
- **Advantages:**
  - Full control over OS and networking
  - Easy to scale (vertically/horizontally)
  - Integrates with load balancers, autoscaling, IAM, and more

EC2 Compute Services

- **Launching EC2 Instances**
  - Choose AMI (Amazon Machine Image):
    - Preconfigured OS (Ubuntu, Amazon Linux, etc.)
  - Choose Instance Type:
    - CPU/RAM (e.g., t2.micro, t3.medium, c5.large)
  - Configure Instance:
    - Number of instances, network settings, IAM role

EC2 Compute Services

- **Add Storage:**
  - EBS (Elastic Block Store) volumes, like a hard drive
- Add Tags
  - e.g., Name=BackendServer, Env=prod
- Configure Security Group
  - Allow inbound/outbound traffic
- Create or Use a Key Pair
  - Used to SSH into the server

EC2 Compute Services

- After launching your instance, you can access it from your terminal using ssh

```
1  ssh -i test.pem ubuntu@3.64.61.98
```

- test.pem is your key pair, ubuntu is your instance username, and the followed by @<your_instance_public_ip_address>

- After a successful connection, you now have ssh access to run commands in your EC2 instance, like cloning your code from github, or running a web server

- To run your application on a specific port, edit the inbound rules of the security group your instance is using to allow access to the port

AWS Elastic IP

- **What is AWS Elastic IP? :**
  - AWS Elastic IP (EIP) is a static, public IPv4 address that you can associate with your AWS resources, like EC2 instances, to ensure they have a consistent public IP address, even if the instance is stopped, restarted, or replaced.
  - This contrasts with public IPv4 addresses automatically assigned to instances, which can change.
  - Mainly used for pointing your domains to a static IPv4 address for consistency

AWS ElasticIP

- **EIP Features:**
  - **Static and Public:**

    Unlike public IPv4 addresses that can change, an Elastic IP remains associated with your AWS account until you explicitly release it, providing a consistent public IP address.

  - **Dynamic Cloud Computing:**

    EIPs are designed to be flexible in cloud environments. You can easily associate and disassociate them from instances as needed.

AWS ElasticIP

- **Failover and Redundancy:**

  If an instance with an EIP fails, you can quickly remap the EIP to a healthy instance within your VPC, minimizing downtime and maintaining service availability.

- **Internet Connectivity:**

  If your instance doesn't have its own public IPv4 address, you can associate an EIP to enable internet connectivity.

AWS ElasticIP

- **DNS and Other Applications:**

  EIPs can be used with DNS records, allowing you to point your domain name to a specific instance, which is useful for websites and other domain-based services.

- **Cost Considerations:**

  AWS charges for EIPs that are allocated but not associated with a running instance, so it's good practice to release EIPs when they are no longer needed.

Lambda Compute Services

- **What Is AWS Lambda (Serverless Compute)?**
  - Lambda lets you run code without managing servers. Just write a function, upload it, and AWS runs it when triggered, you pay only for execution time
  - No provisioning, patching, scaling
  - Supports many runtimes (Node.js, Python, Go, Java, etc.)
  - Use EC2 when you need full OS control or long-lived apps
  - Use Lambda for short, event-driven functions without needing servers

Lambda Compute Services

- **Lambda functions can be triggered by:**
  - API Gateway (HTTP requests)
  - S3 Events (on upload)
  - DynamoDB Streams
  - CloudWatch timers (cron jobs)
- Backend Use Cases:
  - REST API endpoints
  - File processing (image resize)
  - Scheduled backend tasks
  - Real-time processing (IoT, logs, events)

Lambda Compute Services

Lambda VS EC2

Feature	EC2	Lambda
Provisioning	Manual setup (OS, security, scaling)	No provisioning
Use Case	Long-running apps, full control, custom environments	Event-driven, short tasks, fast APIs
Pricing	Pay per running time	Pay per execution time (ms-based)
Scaling	Manual or autoscaling	Automatic
Cold Start?	No	Yes (minor delay on first call)
Max Execution Time	Unlimited	15 minutes max

Lambda Compute Services

- To start using AWS Lambda, head to the aws console and start by creating a node.js function, and set the trigger to API Gateway with HTTP type
- Instal the required modules: **$ npm install @vendia/serverless-express**
- Then upload your project as a .zip file

```
1  const app = express();
2  app.get('/', (req, res) => {
3    res.json({ message: 'Hello from Express on Lambda' });
4  });
5  export const handler = serverlessExpress({ app });
```

AWS Elastic Load Balancing

AWS Load Balancer

- AWS Elastic Load Balancing (ELB) automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses, in multiple Availability Zones. This increases the fault tolerance, availability, and scalability of your applications.

- AWS offers different types of load balancers, each suited for specific use cases:

  - Application Load Balancer (**ALB**)

  - Network Load Balancer (**NLB**)

  - Gateway Load Balancer (**GLB**)

AWS ALB

- **Application Load Balancer (ALB):**
  - Operates at the application layer (Layer 7 of the OSI model).
  - Ideal for HTTP and HTTPS traffic, providing advanced routing features based on URL path, host header, query string parameters, and more.
  - Supports features like WebSockets, HTTP/2, and containerized applications.
  - Used for monolithic and microservices backend applications

AWS NLB

- **Network Load Balancer (NLB):**
  - Operates at the transport layer (Layer 4 of the OSI model).
  - Designed for extreme performance and low latency, handling millions of requests per second.
  - Suitable for TCP, UDP, and TLS traffic where high throughput and static IP addresses are required.
  - Best for real-time applications like messaging applications & online games

AWS GLB

- Gateway Load Balancer (GLB):
  - Operates at Layer 3 (network layer) and Layer 4 (transport layer).
  - Used for deploying, managing, and scaling third-party virtual network appliances like firewalls and intrusion detection systems.
  - Provides a transparent way to insert and chain network appliances.
  - Used in IOT projects

AWS Elastic Load Balancing

- Key Responsibilities
  - Traffic Distribution:
    Routes traffic evenly or based on smart algorithms.
  - Health Checks:
    Detects unhealthy instances and avoids routing traffic to them.
  - Sticky Sessions (Session Persistence):
    Maintains user sessions on the same server if needed.
  - Content-Based Routing:
    Routes requests based on content, e.g., /api vs /images.

AWS Elastic Load Balancing

- **Load Balancing Algorithms**
  - Round Robin: Cycles through available instances.
  - Least Connections: Routes to instance with fewest connections.
  - IP Hash: Routes based on IP address hash (for sticky sessions).
  - Weighted Round Robin: Assigns weights for more powerful servers.

S3 Storage

S3 Storage

- **What Is Amazon S3?**
  - Amazon Simple Storage Service (Amazon S3) is an object storage service offered by Amazon Web Services (AWS).
  - It provides a highly scalable, durable, and available solution for storing and retrieving any amount of data from anywhere on the web.
  - Scalable object storage for any type of file (images, PDFs, videos, backups)

S3 Storage

- **Key characteristics of Amazon S3:**
  - **Object Storage:**
    - S3 stores data as objects within buckets. An object consists of the data file itself and optional metadata describing it. Buckets are the top-level containers for these objects.
  - **Scalability:**
    - It offers industry-leading scalability, allowing users to store and retrieve virtually unlimited amounts of data.

S3 Storage

- **Durability and Availability:**
  - Designed for high durability, ensuring data is protected and accessible even in the event of hardware failures. It also provides high availability for data access.
- **Security:**
  - Offers robust security features, including access control, encryption, and logging, to protect stored data.

S3 Storage

- **Performance:**
  - Delivers high performance for data storage and retrieval, suitable for various applications.
- **Accessibility:**
  - Data can be accessed from anywhere via the internet through a simple web service interface (API) or the AWS Management Console.

S3 Storage

- **S3 Use Cases in Backend Development**

  - User uploads (images, documents, avatars)
  - Static asset storage (PDFs, logs, exports)
  - Backup and restore for databases
  - Serving downloadable content (ebooks, invoices)
  - Storing large media files or analytics exports
  - Often combined with CloudFront for CDN delivery

S3 Storage

- What Are Pre-Signed URLs?
    - A Pre-Signed URL is a secure, temporary link to a specific S3 object, generated by your backend using AWS credentials. It allows anyone with the link to upload or download a file from your S3 bucket — without exposing your AWS credentials or making the bucket public.
- Why Use Pre-Signed URLs?
    - You don’t want your server to process huge file uploads (e.g., images, videos). Instead, your backend generates a URL that allows the frontend to upload directly to S3.
    - You store files in private buckets, but you want users to download them without making the bucket public.

S3 Storage

- **Upload flow:**
  - Frontend requests from backend to upload a file
  - Backend uses AWS SDK to generate a pre-signed PUT URL.
  - Frontend receives the URL and sends a PUT request (with file) directly to S3.
  - S3 stores the file under the given key.
- **Download flow:**
  - Frontend requests from backend to upload a file
  - Backend gives a pre-signed GET URL.
  - Frontend uses that link to download the file securely.

S3 Storage

- To start using AWS S3 in your project, first start by installing the required node modules: **$ npm install @aws-sdk/client-s3 @aws-sdk/s3-request-presigner**

- Then, start by configuring your aws &s3 credentials:

```
export const s3 = new S3Client({
  region: process.env.AWS_REGION,
  credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY_ID,
    secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  },
});
```

S3 Storage

- Create a function to manage uploads to your s3 bucket

```
export const uploadToS3 = async (file) => {
  const key = `${Date.now()}-${file.originalname}`;

  const params = {
    Bucket: process.env.S3_BUCKET,
    Key: key,
    Body: file.buffer,
    ContentType: file.mimetype,
  };

  const command = new PutObjectCommand(params);
  await s3.send(command);

  return key;
};
```

S3 Storage

- Finally, add a function to generate presigned urls for accessing data:

```
export const generatePresignedUrl = async (key) => {
  const command = new GetObjectCommand({
    Bucket: process.env.S3_BUCKET,
    Key: key,
  });

  const url = await getSignedUrl(s3, command, { expiresIn: 60 * 5 });
  return url;
};
```

S3 Storage

- You can now use multer to upload files to s3 in your controller:

```
router.post("/upload", upload.single("image"), async (req, res) => {
  try {
    const key = await uploadToS3(req.file);
    const url = await generatePresignedUrl(key);
    res.json({ success: true, key, url });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Upload failed", details: err.message });
  }
});
```

RDS & DynamoDB For Databases

AWS Databases

- **AWS Offers Two Broad Database Categories:**

    1. For Relational (SQL) databases we use Amazon RDS for a structured, schema based approach

    2. For non-relational databases (NoSQL) databases we use Amazon DynamoDB for a key-value or document store, schema-less approach

RDS for: MySQL,Postgreql, etc.

DynamoDB for: key-value document store

AWS RDS

- **What is AWS RDS?:**
  - Amazon Relational Database Service (Amazon RDS) is a managed service provided by Amazon Web Services that simplifies the setup, operation, and scaling of a relational database in the cloud.
  - It manages common database administration tasks, such as provisioning hardware, patching, backups, and scaling, freeing users to focus on application development and business logic.

AWS RDS

- **Key features and benefits of Amazon RDS:**
  - **Managed Service:**

    AWS handles the underlying infrastructure and administrative tasks, reducing operational overhead.

  - **Database Engine Support:**

    It supports various popular relational database engines, including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, SQL Server, and Db2.

AWS RDS

- **Scalability:**

  Users can easily scale compute and storage resources up or down to meet changing demands.

- **High Availability and Durability:**

  Features like Multi-AZ deployments and automated backups enhance database availability and data durability.

- **Cost-Effective:**

  It offers a pay-as-you-go pricing model with no upfront investments, and users only pay for the resources consumed.

AWS RDS

- **Security:**

  Amazon RDS provides various security features, including network isolation, encryption at rest and in transit, and integration with AWS Identity and Access Management (IAM).

- **Monitoring and Metrics:**

  Integration with Amazon CloudWatch allows for comprehensive monitoring of database performance and resource utilization.

AWS RDS

- **How to create an RDS database? :**
  - Select Engine: MySQL or PostgreSQL
  - Choose DB instance size (e.g., db.t3.micro for free tier)
  - Connect it to your EC2 instance
  - Set your database credentials
  - **Add inbound rule in the RDS security group to allow:**
    - Port 3306 for MySQL
    - Port 5432 for PostgreSQL
    - From EC2 security group or your IP

AWS RDS

- Now in your node application, to connect to your RDS instance, start by installing the required modules and configuring your remote database: **$ npm install pg** **// for postgresql**

```
import pkg from 'pg';
import dotenv from 'dotenv';

dotenv.config();

const { Pool } = pkg;

const pool = new Pool({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  ssl: {
    rejectUnauthorized: process.env.SSL_REJECT_UNAUTHORIZED === 'true'
  }
});
```

AWS RDS

- You can now start querying your database using SQL queries or ORMs using this pool object:

```
app.get('/ids', async (req, res) => {
  try {
    const result = await pool.query('SELECT id FROM testtable');
    res.json(result.rows);
  } catch (error) {
    console.error('Error fetching IDs:', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
});
```

AWS RDS

Notes:

1. Your ec2 instance must have a psql-client package to connect to your database

   $ sudo apt install psql-client –y

2. Never hardcode your database credentials in your code, use a .env file

3. Make sure that your database security group inbound rules allow connections from your ec2 instance

4. If you want to access your RDS from your local machine, make sure to add your ip address to the inbound rules of your RDS security group

AWS DynamoDB

- What is AWS DynamoDB:
  - Amazon DynamoDB is a fully managed, serverless NoSQL database service offered by Amazon Web Services.
  - It is designed to provide fast and predictable performance with seamless scalability for applications requiring high throughput and low latency.
  - DynamoDB is well-suited for a wide range of applications, including web, mobile, gaming, IoT, and ad tech, where high performance, scalability, and availability are critical.

AWS DynamoDB

- Key characteristics and features of Amazon DynamoDB:
  - NoSQL Database:

    DynamoDB is a non-relational database, supporting key-value and document data models. It does not use SQL for querying but instead provides a proprietary API accessible through AWS SDKs in various programming languages.

  - Fully Managed and Serverless:

    AWS handles all the underlying infrastructure, including server provisioning, patching, backups, and scaling. Users only pay for the resources consumed (read/write capacity units and storage).

AWS DynamoDB

- High Performance and Scalability:

  DynamoDB delivers single-digit millisecond response times at any scale. It automatically scales to handle any amount of data and request traffic, ensuring consistent performance.

- Durability and High Availability:

  It is designed for high durability and offers features like global tables for multi-region, multi-master replication to enhance availability and disaster recovery.

AWS DynamoDB

- Built-in Features:

  DynamoDB includes integrated features such as point-in-time recovery for continuous backups, in-memory caching with DynamoDB Accelerator (DAX) for faster reads, and integration with other AWS services like Kinesis for real-time data streams.

- Flexible Data Model:

  It supports schemaless data, allowing for flexible evolution of data structures without rigid schema definitions.

AWS DynamoDB

- **How to create your DynamoDB table? :**
  - First start by creating the table by defining the table name (eg. Books), then the partition key (eg. bookId), which will be your primary key for your table, finally the sort key (eg. chapter), the sort key allows you to sort or search among all items sharing the same partition key.
  - Then start by configuring your aws credentials and DynamoDB in your app:

```
AWS.config.update({
  region: process.env.AWS_REGION,
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY
});
const dynamoDb = new AWS.DynamoDB.DocumentClient();
```

AWS DynamoDB

You can now start implementing CRUD operations on your DynamoDB table:

```
app.post('/books', async (req, res) => {
  const book = {
    bookId: Date.now().toString(),
    title: req.body.title,
    author: req.body.author,
    chapter: req.body.chapter
  };

  try {
    await dynamoDb.put({
      TableName: BOOKS_TABLE,
      Item: book
    }).promise();

    res.json(book);
  } catch (err) {
    res.status(500).json({ error: 'DynamoDB error', details: err.message });
  }
});
```

CloudWatch

CloudWatch

- **What is AWS CloudWatch? :**
  - AWS CloudWatch is a monitoring and observability service for AWS resources, applications, and services.
  - It provides a unified view of operational health by collecting and tracking metrics, monitoring log files, setting alarms, and enabling automated actions.
  - CloudWatch helps users gain insights into resource utilization, application performance, and overall operational health, allowing them to optimize performance, troubleshoot issues, and ensure smooth application operation.

CloudWatch

- **Key features:**
  - **Monitoring Resources and Applications:**

    CloudWatch monitors various AWS resources like EC2 instances, DynamoDB tables, RDS DB instances, and also custom metrics from applications and services.

  - **Collecting and Tracking Metrics:**

    CloudWatch gathers metrics (numerical data) about resource performance, such as CPU utilization, network traffic, and latency, and makes them available for analysis and visualization.

CloudWatch

- **Monitoring Log Files:**

  CloudWatch Logs allows users to collect, store, and monitor log files from various sources, including EC2 instances, AWS CloudTrail, and Route 53.

- **Setting Alarms:**

  Users can define alarms based on specific metrics or log data patterns. These alarms can trigger notifications or automated actions when certain thresholds are breached.

CloudWatch

- **Automated Actions:**

  CloudWatch can automatically respond to alarms by scaling resources, rebooting instances, or performing other predefined actions to maintain optimal performance and availability.

- **Visualizing Data:**

  CloudWatch provides dashboards and visualizations to help users analyze performance data, identify trends, and troubleshoot issues.

CloudWatch

- **In essence, CloudWatch empowers users to:**
  - Gain insights into the health and performance of their AWS resources and applications.
  - Proactively identify and address potential issues before they impact users.
  - Optimize resource utilization and reduce costs.
  - Improve application availability and performance.
  - Automate operational tasks and streamline workflows.

CloudWatch

- **How to use CloudWatch Agent in your ec2 instance? :**
  - Install CloudWatch Agent on EC2

```
wget https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb
sudo dpkg -i amazon-cloudwatch-agent.deb
```

  - Configure /etc/awslogs/awslogs.conf

```
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard
```

  - Logs are pushed to CloudWatch Log Groups

CloudWatch

- **How to set CloudWatch alarms? :**
  - Navigate to CloudWatch: Open the AWS Management Console and go to the CloudWatch service.
  - Create Alarm: In the navigation pane, choose "Alarms," then "All alarms," and click "Create Alarm."
  - Select Metric:
    - Click "Select metric."
    - Choose "EC2" from the list of services.
    - Select "Per-Instance Metrics" and locate your specific EC2 Ubuntu instance.
    - Choose the desired metric to monitor (e.g., CPU Utilization, NetworkIn, DiskReadBytes).

CloudWatch

- Configure Metric and Conditions:
  - Define the statistic (e.g., Average, Sum, Maximum) and period (e.g., 5 minutes, 1 minute) for the metric.
  - Set the threshold and comparison operator (e.g., "Greater than," "Less than").
  - Specify the number of evaluation periods and datapoints within those periods that must breach the threshold to trigger the alarm.

CloudWatch

- **Configure Actions:**
  - Choose the action to take when the alarm state is triggered. This can include:
  - Sending notifications: Select an existing Amazon SNS topic or create a new one to send email or other notifications.
  - EC2 actions: Automatically stop, terminate, reboot, or recover the instance.
- **Name and Describe:** Provide a descriptive name and an optional description for your alarm.
- **Create Alarm:** Review the alarm configuration and click "Create alarm."

AWS SNS

- **What Is SNS (Simple Notification Service) ?:**
  - A fully managed pub/sub messaging service.
- **Use Cases:**
  - Send email/SMS when alarm triggers
  - Notify developers on failure
  - Trigger Lambda functions
- Flow Example:

CloudWatch Alarm → SNS Topic → Email Subscribers

DevOps CI/CD Pipeline in AWS

CI/CD on AWS

- CI/CD (Continuous Integration & Continuous Deployment) automates the process of building, testing, and deploying code.
- CI: Automatically test code on every push/merge (ensures quality)
- CD: Automatically deploy code to production after passing tests
- AWS CI/CD Stack:
    - CodeCommit (Git hosting, optional)
    - CodeBuild (build & test)
    - CodeDeploy (deployment service)
    - CodePipeline (orchestrates the full workflow)

AWS CodePipeline

- What is AWS CodePipeline? :
  - AWS CodePipeline is a continuous delivery service provided by Amazon Web Services (AWS) that automates the release process for software and infrastructure updates.
  - It allows users to model, visualize, and automate the various stages involved in releasing software, from source code changes to deployment in production environments.

AWS CodePipeline

- Key features and concepts of AWS CodePipeline:
  - **Continuous Delivery:**

    CodePipeline facilitates continuous delivery by automating the build, test, and deployment phases of the software release process.

  - **Stages and Actions:**

    A pipeline is composed of stages, which are logical units representing a phase in the release process (e.g., Source, Build, Test, Deploy). Each stage contains actions, which are specific tasks performed within that stage (e.g., compiling code, running tests, deploying to an environment).

AWS CodePipeline

- **Integration with AWS Services:**

  CodePipeline integrates seamlessly with other AWS developer tools like AWS CodeCommit (source control), AWS CodeBuild (build and test service), and AWS CodeDeploy (deployment service). It also supports integration with third-party tools.

- **Automation:**

  CodePipeline automatically triggers pipeline executions in response to code changes in the source repository, ensuring that new changes are consistently built, tested, and deployed according to the defined workflow.

AWS CodePipeline

- **Visualization:**

  It provides a visual representation of the pipeline, allowing users to monitor the progress of releases and identify any issues or bottlenecks.

- **Artifacts:**

  Artifacts are the output of actions within a stage and serve as input for subsequent actions or stages (e.g., compiled code, test reports).

AWS CodePipeline

- How to create a CI/CD pipeline using AWS CodePipeline:
    - First start by choosing "Build custom pipeline"
    - Choose a queued execution mode and define your service roles
    - Then select your source provider (which repo to monitor for changes) and connect to it, this is the first stage that our pipeline will got through
    - Choose your default branch default branch
    - Then in the deploy stage select your deploy Provider as AWS EC2 and choose your ec2 instance
    - Type your target directory as an absolute path (eg. home/ubuntu/my-api)
    - Then type the location of your post script as a relative path (eg. Scripts/post-deploy.sh), this script is a bash script thet will execute a ist of commands after sourcing our repo

Any Questions