AWS:

(Took an AWS certification on Friday so notes are mainly from studying for that)
Operation Excellence:

Organization - Cloud infrastructure setup and organization
Prepare - Assessments and audits
Operate - Track and manage operations
Evolve - Service Analytics

CloudFormation

Performance Efficiency

Selection - Create the correct solution for the task, use AWS features
Review - Keep up to date with AWS
Monitoring - Monitor performance
Tradeoffs - Analyze pros and cons of approaches and which fits best

CloudWatch

Security

Security - Configurations/Understanding
Identity and Access Management - Accounts and Permissions
Detective Controls - Logging and Log Management
Infrastructure Protection - Networking security
Data Protection - Encryption, Data Storage, Key Management
Incident Response - Logging

IAM

Reliability

Foundations - Track, assess, and prevent disruptions
Change Management - Expect and respond to change automatically
Failure Management - Prevent and be able to quickly respond to failures
Workload Architecture - Design systems and use technologies that are reusable and scalable

CloudWatch

Sustainability

Region Selection - Select best region for business uses
User Behavior Patterns - Increase and decrease resources according to user demand
Software and Architecture Patterns - Optimize software and infrastructure to increase resource utilization and and efficiency
Data Patterns - Optimize how data is accessed and what data needs to be stored
Hardware Patterns - Select correct hardware types and resources for the tasks needed
Development and Deployment Process - use strategies to quickly provision resources when you need them

EC2 AutoScaling

Cost Optimization

Cloud Financial Management - Cost tracking and assessments
Cost-Effective Resources - Cheaper Options for same workloads, cost assessments
Matching Supply and Demand - Scaling resources correctly
Expenditure Awareness - Cost tracking and budgeting
Optimizing Over Time - Keep up to date and monitor what is currently being used

Cost Explorer

Service To Review:

CloudWatch
CloudTrail
S3
SQS
SNS
IAM
FSx
OpsWorks
Global Accelerator
Storage Gateway

SQS:

Messages contain up to 256 KB of text that can be stored from 1 to 14 days
Messages in Queue are encrypted server side with KMS keys
Standard vs FIFO
Standard messages can be sent in any order and may occasionally repeated
FIFO will deliver messages exactly once in the order they are sent with with a max of 300 messages / second (or 3000 / second if batch the max of 10 at a time)
Dead-letter queues can store messages that reach the max processing attempts
Visibility timeout helps prevent multiple receivers from processing the same message, default is 30 seconds, max is 12 hours
Short polling vs long polling

SNS:

Publishers and subscribers to a topic
Can encrypt topics with customer master keys
SNS can transport via Email, HTTP/S, SMS, SQS, or Lambda
Used for alerts, push messages, or notifications

Recovery Point Objective: RPO, maximum acceptable amount of data loss measured in time
Recovery Time Objective (RTO): How much down time after disaster is acceptable

IAM:

Can attach policies to identities or resources
Identities: Actions that may or may not be performed by a user, group, or role

Web App Infrastructure:

- Host server hosts the backend app
- Client side front end communicates to that back end
- User > web server > web app server > data storage
- Communicate over HTTP
- Pages usually rendered on client side using HTML/CSS/JS
- Front end and back end communicate through HTTP requests

Elastic Beanstalk:

- AWS managed services that automatically manages application deployment infrastructure
- Handles: Configuration, deployment, load balancing, auto scaling, health monitoring, debugging, and logging
- Upload code and handles automatic deployment on servers
- Supports all Python, Go, and Docker
- No extra charge, just pay for resources used
- https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/go-tutorial.html

Database Options:

MySQL:
- Most commonly use SQL DBMS
- Purely relational databases
- Relatively simple
- Support in both Python and GoLang

PostgreSQL:
- SQL DBMS with added object functionality (ORDBMS)
- Supports synchronous data replication
- More complicated but provides more and extensible functionality

MariaDB:
- Forked version of MySQL
- Supports most MySQL functionality but can be more performant
- Can be more suitable for smaller databases compared to other DBMSs

Database/SQL refresh:
- Primary key: key used to identify each individual item in a table

- Foreign key: Primary key from another table within a table
- Unions combine results from multiple selects
- Joins essentially combine data from multiple tables before query
- Create, drop, update tables
- Can auto increment ids with MySQL, although should probably hash and use again if using it for things like Paths

Go Databases:

- To use SQL databases in Go apps you need to to both:
    - Import “database/sql” for interface and objects for interacting with the DB in code
    - Import DBMS specific driver, implements JDBC/ODBC

Go WebApps:

- “net/http” package is used to build web applications
- “html/template” library can process static html templates
- https://pkg.go.dev/html/template
- Can create REST APIs with Gin framework

Dependency Management:

- Go manages dependencies using go.mod files.
- Go checks for dependency differences between import statements and the mod file
- Can set custom versions or proxies for packages between multiple people in an organization
- Poetry can manage dependencies for python environments between multiple team members
- Poetry can also build and public python projects

Django:

- Most popular Python web framework
- Supports build in auth and account management features
- Can be complicated to use, essentially have to learn an entire framework that kinda happens to use Python
- Supports front end development in the same framework, most apps organized through a model view controller

Python MySQL:

- Install and import mysql.connector driver
- Can use mysql.connector.connect(user, host, db), etc.
- Can also set a db config dictionary
- To use with Django need to update settings.py DATABASES

3rd Party Auth Services:
Amazon Cognito:

- Manages user sign-up, sign-in, and an access control for websites
- User pools create sign-in and sign-up, social media sign in, a built in UI, MFA, and user directory management
- Can be managed/used through AWS SDK
- After successful user pool authentication, the app receives user token
- Uses OAuth2 authentication flows in applications
- https://oauth.net/2/

Microservices:

- Create rest API for separate services to communicate with each other and with front end
- Typically used with containers to separate services and scale individually
- Use standard templating/formatting to communicate between services, typically done with something like JSON
- Each service typically uses it own database, with this services stay loosely coupled and different services can use different types of databases as needed

Containerization:

- Containers offer a form of virtualization that allows different apps/services to be separated but be able to run on the same hardware server
- Supports separated scaling of services by being able to pass a docker build file to a new instance that will run it with the same settings as the rest of the services
- Containers typically designed and run with Docker
- Can manage multiple Docker instances with orchestration tools such as Kubernetes