**Questions (30-60 min):**  
  
General:

* Can you tell me briefly about some projects you've worked on and the approach you took from start to finish?
* Can you tell me the approach, from start to finish about a Problem You Faced/Solved?
* What programming languages or frameworks are you familiar with? Can you share examples/experience with them?
* What Cloud Platforms do you have experience with? What were the main areas of focus in the platform?

Frontend Development:

* Can you describe your workflow when you create a web page?
* Can you explain some techniques you have used to increase performance?
* Can you in your own words describe Cross-Site Scripting (XSS) attacks and how to prevent them? (Hint using CSP Header)
* How can you ensure that your website design or web application is accessible and user-friendly?
* Can you describe what `lazy loading` is?

Angular:

* How are Angular Components used to create a User Web Interface?
* When loading website data, how do asynchronous calls and lifecycle hooks play a role?
* Can you describe what data binding is in the Angular framework? (A: 2-way Binding)
* What are some advantages of Angular over other front-end frameworks?
* What are some approaches to communicating with Server Backend?

RxJS:

* Can you explain the difference between Promise and Observable in Angular?
* What is RxJS? How is it used for Angular Applications?
* What are the core features of RxJS?
* Can you describe the difference between the tap, map, and switchMap operators?
* When should we use the forkjoin, switchMap, mergeMap, exhaustMap, and concatMap in RxJS?

Microservices Development:

* Can you describe the main components and features of Microservices?
* What principles/best practices are used when designing Microservices?
* What challenges do people face while working with Microservice Architectures?
* What is the role of Idempotence in Microservices?

Golang:

* TODO: Add Questions

DevOps:

* In your own words, what is Configuration Management, Continuous Integration, Continuous Testing, and Continuous Deployment?
* What DevOps Tools have you worked with? Can you share an example?
* How do DevOps concepts/tools work together? Starting from local development all the way to deployment of an application?

K8s/Containerization:

* What are containers? What are the core concepts of containerization?
* Can you explain the basic concepts of container-orchestration and deployment?
* Can you explain what role container-orchestration has with Cloud Services?

Bonus Questions:

* How do you assure Software Quality
* Can you compare Cloud and On-Premise Computing?
* Can you describe a container's lifecycle?
* In your own words could you describe Kubernetes?
* How is Kubernetes related to containers?
* Can you explain the difference between deploying applications on hosts and containers?

**Fun Puzzle Problem Question (10-15 min):**

*You have 8 pennies, 7 weigh the same, one weighs less. You also have a judges scale. Find the one that weighs less (in the least amount of steps).*

1. **8 Steps** - Brute Force (weigh each penny against each other) - O(n)
2. **3 Steps** - Binary Search (divide pennies in half each time) - O(log n)
   1. *log n* is base 2 due to division of 2
3. **2 Steps** - Custom Algorithm (Two Possible Outcomes) - O(log n)

Split the 8 pennies into 3 groups of 3,3,2 pennies. Weigh the first two groups of 3

pennies each.

1. They weigh the same. Therefore, take third group of 2 pennies and find the

lighter coin.

2. Group 1 weighs more than Group 2. Take group 2 (3 pennies) and pick any 2

out of 3. If they weigh the same, then the third penny is lighter.

**Answers:**

* XSS
  + XSS users executed malicious scripts (also called payloads) unintentionally by clicking on untrusted links and hence
  + Use Content Security Policy Header, specify Content-Type Headers
* Lazy Loading
  + Lazy loading is a design consideration that delays the initialization of any object until it is required. When used correctly, it can enhance website or application responsiveness and boost performance
* How are Angular Components used to create a User Web Interface?
  + Components are the basic building blocks of the user interface of an Angular application. Every component is associated with a template and is a subset of directives. An Angular application typically consists of a root component, which is the AppComponent, that then branches out into other components creating a hierarchy
* Angular Lifecycle Hooks
  + ngOnChanges() - called when Angular resets data bound properties.
  + ngOnInit() - used to initialize the directive or component and sets data bound properties.
  + ngDoCheck() - when the changes are not detected by angular own then it's used to catch those changes.
  + ngAfterContentInit()
  + ngAfterContentChecked()
  + ngAfterViewInit()
  + ngAfterViewChecked()
  + ngOnDestroy() - used to cleanup the component or directive before Angular destroys it.
* Angular Binding
  + Angular uses two-way binding. Any changes made to the user interface are reflected in the corresponding model state. Conversely, any changes in the model state are reflected in the UI state. This allows the framework to connect the DOM to the Model data via the controller. However, this approach affects performance since every change in the DOM has to be tracked.
* What are some advantages of Angular over other front-end frameworks?
  + MVC architecture - Angular is a full-fledged MVC framework. It provides a firm opinion on how the application should be structured. It also offers bi-directional data flow and updates the real DOM.
  + Modules: Angular consists of different designs patterns like components, directives, pipes, and services, which help in the smooth creation of applications.
  + Dependency Injection: Components dependent on other components can be easily worked around using this feature
  + Other generic advantages include clean and maintainable code, routerModule, unit testing, reusable components, data binding, and excellent responsive experience
* What are some approaches to communicating with a Server Backend (ex: proxy, k8s)
  + Proxy backend server, use proxy.conf.json in root of Angular CLI project
    - Useful for local development
  + Host dist files on web server or backend able to serve host files
  + Kubernetes Deployment, declare services allowed to communicate with each other
    - ex: Service for Frontend and Backend
    - Need to containerize both frontend and backend
    - Further Answer:
      * Use K8 and API gateway such as Emissary/Ambassador for handling inbound and outbound traffic in Kubernetes cluster
* Can you explain the difference between a Promise and Observable in Angular?
  + **Promises:**
    - return a single value
    - not cancellable
    - more readable code with try/catch and async/await
  + **Observables:**
    - work with multiple values over time
    - cancellable
    - support map, filter, reduce, and similar operators
    - use Reactive Extensions (RxJS)
    - an array whose items arrive asynchronous over time
* What is RxJS? How is it used for Angular Applications?
  + Javascript library that uses observables to work with reactive programming and deals with asynchronous data calls, callbacks, and event-based programs.
  + RxJS has introduced the concept of "reactive programming" to the web. It implements a reactive extension for Typescript and Javascript
  + RxJS works as a combination of the observer pattern, iterator pattern and functional programming
* What are the core features of RxJS?
  + **Observer**
    - The Observer is an object with next(), error(), and complete() methods, which are called when we have to interact with the observable, i.e., the source interacts for an example button click, http request, etc
  + **Observable**
    - In RxJS, an observable function is used to create an observer and attaches it to the source where values are expected. For example, clicks, mouse events from a DOM element or a Http request, etc
  + **Subscription**
    - The role of the subscription comes in the scene when the observable is created. To execute the observable, we need to subscribe to it. It can also be used to cancel the execution
  + **Operators**
    - Operators are a very important part of RxJS. An operator is a pure function that takes observable input and emits the result in the output form. Input and output both are observable.
  + **Subject**
    - A subject is a observable that can multicast, i.e., talk to many observers. Suppose we have a button with an event listener. The function attached to the event using addlistener is called every time the user clicks on the button. Similar functionality goes for the subject too.
  + **Sechedulers**
    - A scheduler controls the execution of when the subscription has to start and be notified.
* Can you describe the difference between tap, map, and switchMap operators?
  + Tap
    - Side effects inside a steam. So this operator can be used to do something inside a stream and return the same observable as it was used on. It runs a method to emit a plain isolated side effect.
  + Map
    - Takes the value from a steam, can manipulate it and passes the manipulated value further to the steam again.
  + SwitchMap
    - It is kind of a map operator, but it should resolve the first observable first, use the values and then switch to the next observable while keeping the steam. So that when we subscribe we get to the (real) values of the last observable.
* When should we use the forkoin, switchMap, mergeMap, exhaustMap, and concatMap in RxJS?
  + **forkoin** - for a group of source items, waits for all observables to complete and emit final values
  + **mergeMap** - (ordering by inner observable) - creates an observable immediately from any source item, all previous observables are kept alive
  + **concatMap** - (ordering by outer observable) - waits for the previous observable to complete before creating the next one
  + **switchMap** - (throttle by last inner observable) - for any source item, completes the previous observable and immediately creates the next one (only cares about last value sent)
  + **exhaustMap** - (throttle by first inner observable) - source items are ignored while the previous observable is not completed
  + **Simplified Explanation**
    - forkjoin - make parallel calls
    - concapMap - makes the call in a serial fashion
    - mergeMap - makes the call in a parallel fashion
    - switchMap - makes the call, but then if a new request comes in ignores (unsubscribes) the previous call
    - exhaustMap - makes the call and ignores any new request until the previous call comes back
* Can you explain the importance of API Gateways for Kubernetes and any examples?
  + An API Gateway is an API Management tool that sits between a client and a collection of backend services
  + An API Gateway acts as a reverse proxy to accept all application programming interface (API) calls, aggregate the various services required to fulfill them, and return the appropriate result.
    - You want to protect your APIs from overuse and abuse, you you use an authentication service and rate limiting
    - You want to understand how people use your APIs, so you've added analytics and monitoring tools
    - If you have monetized APIS, you'll want to connect to a billing system
    - You may have adopted a microservices architecture, in which case a single request could require calls to dozens of distinct applications
    - Over time you'll add some new API services and retire others, but your clients will still want to find all your services in the same place
* Examples of K8s API Gateways: Kong, Ambassador, Gloo, Istio