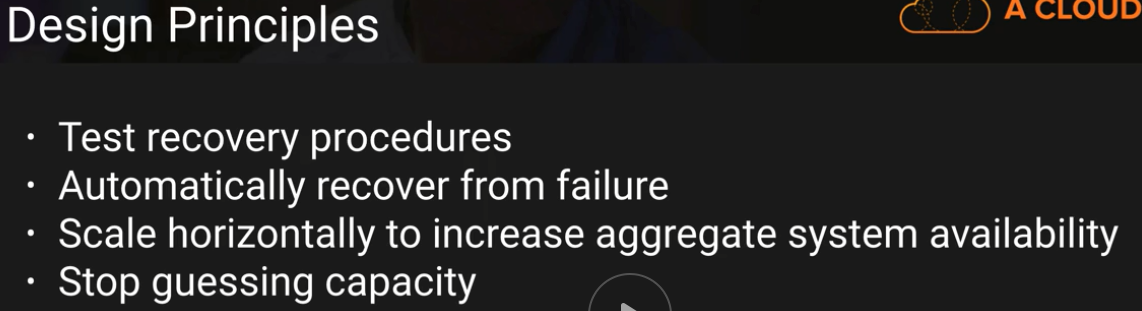
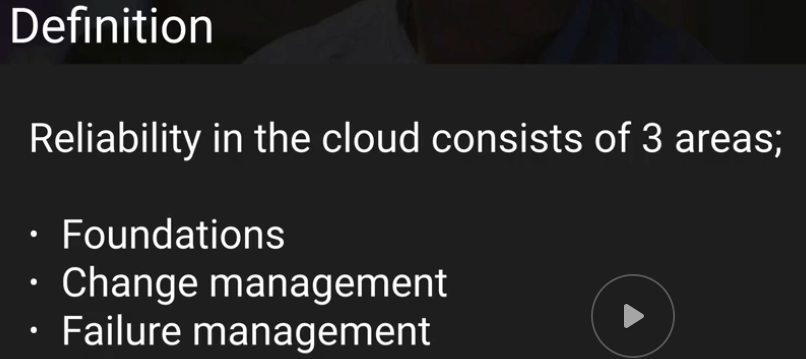
Reliability

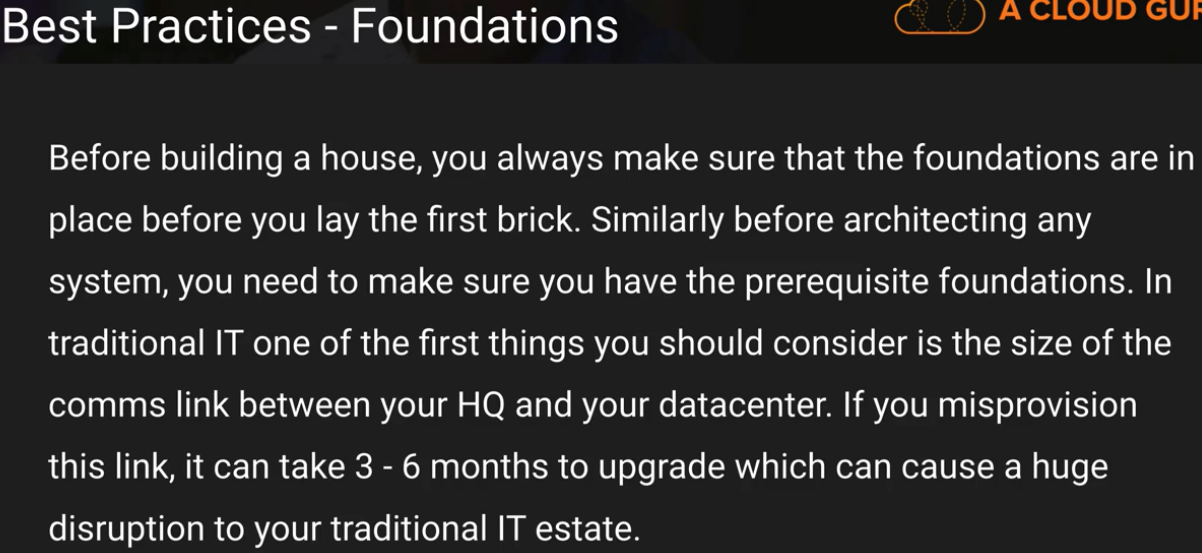
Ability to recover from failures and load…

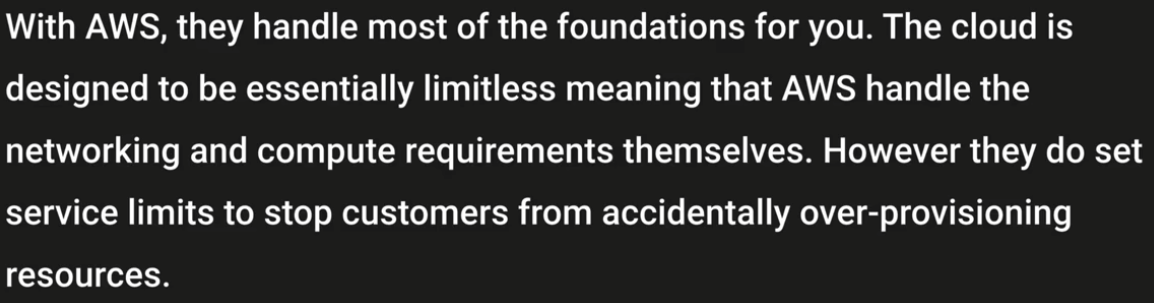
Failures ?  Recover

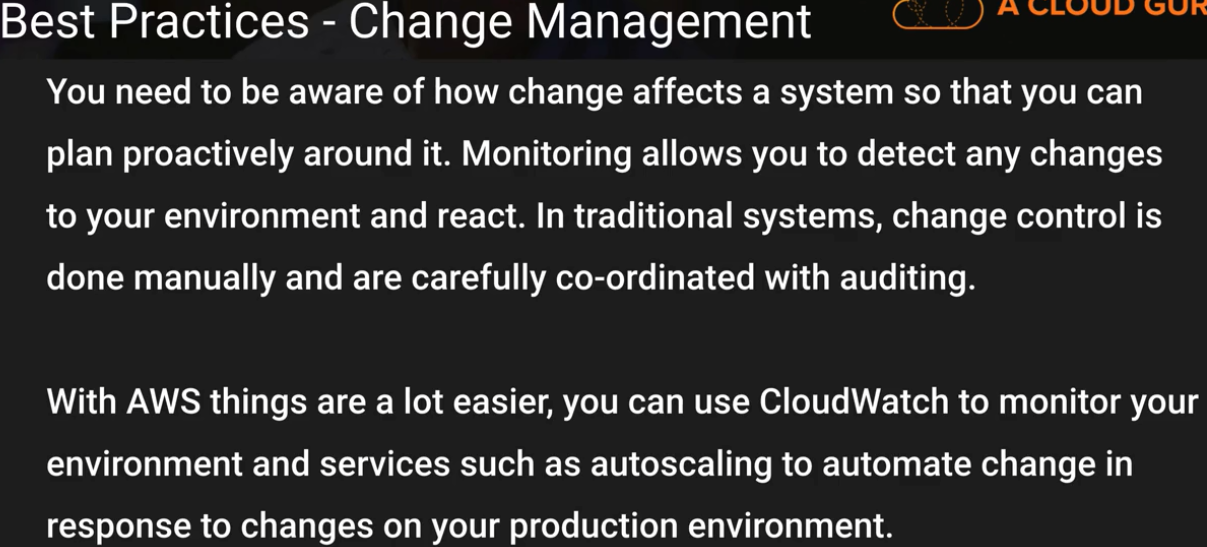
Guess capacity – problem –> end up overprovision or underprovision

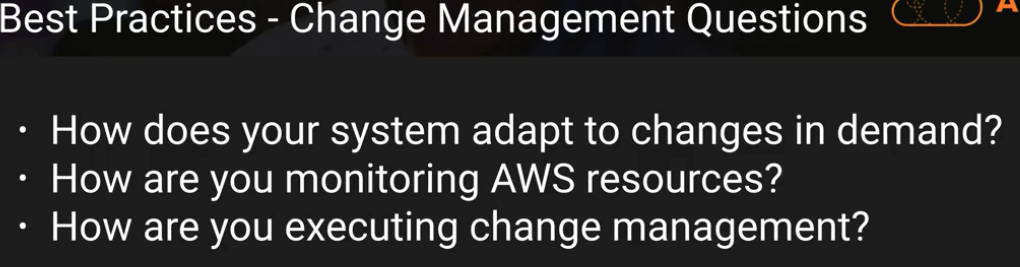


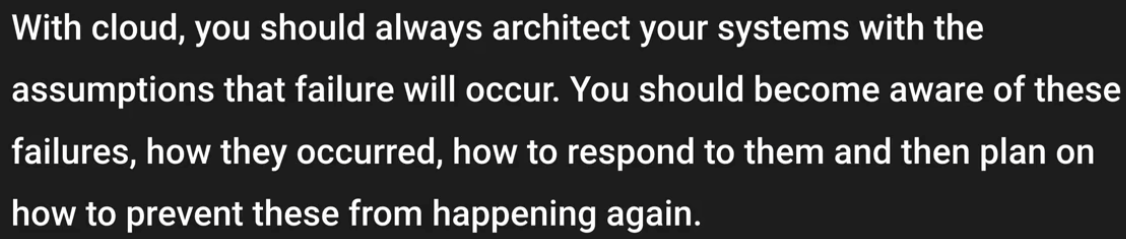


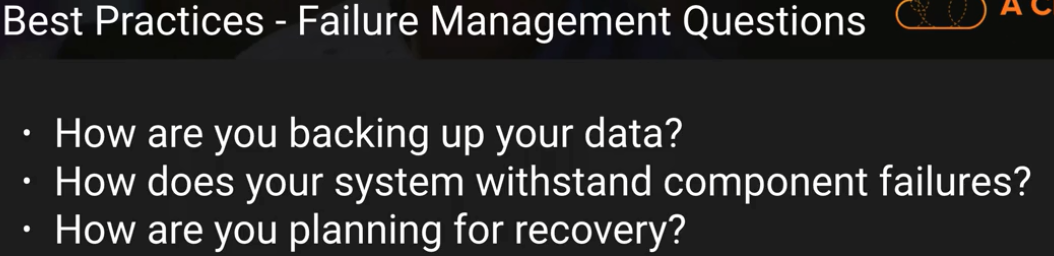


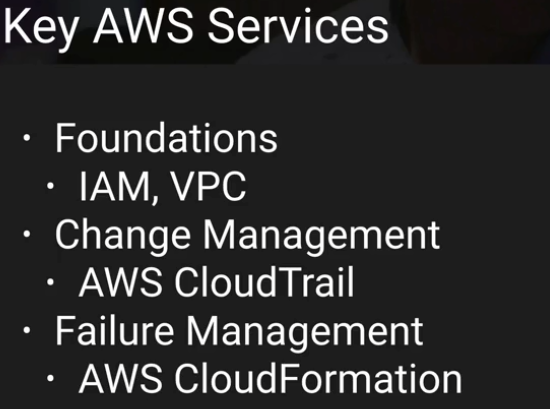












Thanks & Regards,

Shantaram Vernekar

Reliability

* ability of a system to recover from infrastructure or service disruptions,
* dynamically acquire computing resources to meet demand,
* mitigate disruptions such as misconfigurations or transient network issues

Definitions

*Service availability* is commonly defined as the percentage of time that an application is operating normally

Availability = Normal Operation Time / Total Time

Time – normally over a year

99.99

**Calculating availability with hard dependencies**

99.99% (application) \* 99.99% (downstream 1) \* 99.99% (downstream 2)

* **99.97 %**

**Calculating availability with redundant components**

100% - (0.1% \* 0.1%)

* **99.999%**

**Calculating dependency availability.**

determine the Mean Time Between Failure (MTBF) and Mean Time to Recover (MTTR). An availability estimate can be established by:

Availability = MTBF / (MTBF + MTTR)

For example, if the MTBF is 150 days and the MTTR is 1 hour, the availability estimate is 99.97%.

**Costs for availability.**

Foundation – Networking

**Large CIDR block – size can be reduced later**

if you choose to connect your VPC to your data center via an AWS Direct Connect connection, you should have a redundant connection fallback either through a second Direct Connect connection from another provider or through the internet

**Key Services for Network Topology**

**VPC**

**Others:**

**AWS Direct Connect**:

**Amazon EC2**:

**Amazon Route 53**:

**Elastic Load Balancing**:

**AWS Shield**:

Application Design for High Availability

When designing a new application, it’s common to assume that it must be “five nines” (99.999%)

most internet service providers aren’t built to achieve five nines of availability. Therefore, multiple service providers (with no common point of failure) are required for the application to be 99.999% available to a specific end custome

The whole system will require exhaustive testing for failure triggers.

Because 99.999% availability provides for less than 5 minutes of downtime per year,

With a 5 minute per year budget, human judgment and action is completely off the table for failure recovery. The system must automatically recover under every situation.

Applications that truly require 99.999% availability can be built on AWS,

|  |  |
| --- | --- |
| The following table list common sources of interruption: **Category** | **Description** |
| **Hardware failure** | Failure of any hardware component in the system, including in hosts, storage, network, or elsewhere. |
| **Deployment failure** | Failure caused directly as a result of a software, hardware, network, or configuration deployment. This includes both automated and manual changes. The rest of the buckets specifically do not meet this definition. |
| **Load induced** | Load related failures can be triggered by a change in behavior, either of a specific caller or in the aggregate, or by the service reaching a tipping point. Load failures can occur in the network. |
| **Data induced** | An input or entry is accepted by the system that it can’t process (“poison pill”) |
| **Credential expiration** | Failure caused by the expiration of a certificate or credential. |
| **Dependency** | Failure of a dependent service results in failure of the monitored service. |
| **Infrastructure** | Power supply or environmental condition failure has an impact on hardware availability. |
| **Identifier exhaustion** | Exceeding available capacity, a throttling limit was hit, an ID ran out, or a resource that is vended to customers is no longer available |

Achieving 99.999% availability means mastering all of the sources of interruption listed here and automating all human intervention out of operational processes.

Understanding Availability Needs

upon closer inspection we frequently find that certain aspects of an application or service have different availability requirements.

Services might have very high availability requirements during certain hours of the day, but can tolerate much longer periods of disruption outside of these hours

These are a few of the ways that you can decompose a single application into constituent parts, and evaluate the availability requirements for each

The benefit of doing so is to focus efforts (and expense) on availability according to specific needs, rather than engineering the whole system to the strictest requirement

divide services into the “data plane” and the “control plane.”

Data Place – Read/Write operations to DB

Control Plane – Adding EC2 instance, changing table metadata

the data planes typically have higher availability design goals than the control planes.

Naturally, components that have higher availability design goals will necessitate deeper investment in the engineering, testing, and operations automation

Application Design for Availability

• Operational Considerations for Availability

• Example Implementations for Availability Goals

Thanks & Regards,

Shantaram Vernekar

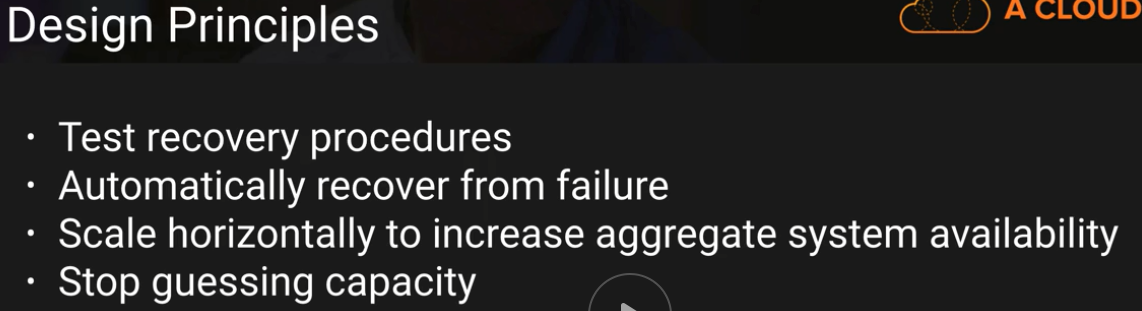
**From:** Shantaram Vernekar   
**Sent:** Sunday, June 24, 2018 6:09 PM  
**To:** Shantaram Vernekar <shantaram\_vernekar@persistent.com>  
**Subject:** AWS - Well Architected framework - Reliability

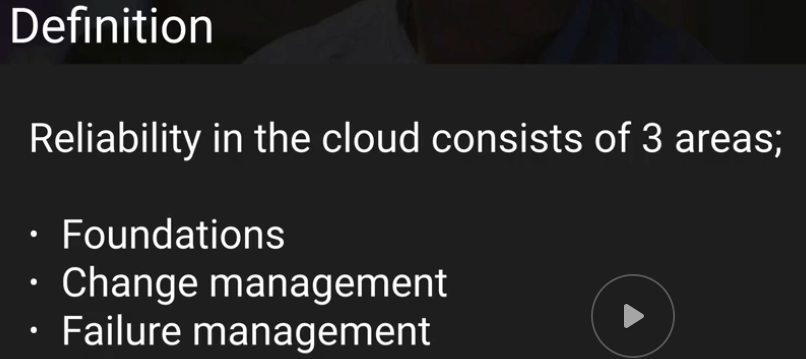
Reliability

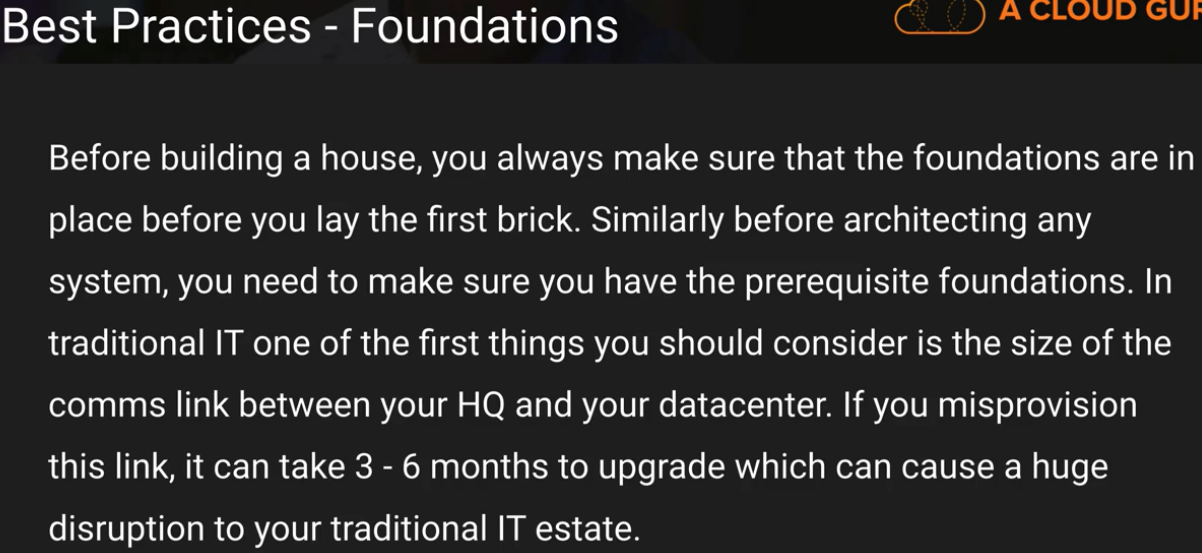
Ability to recover from failures and load…

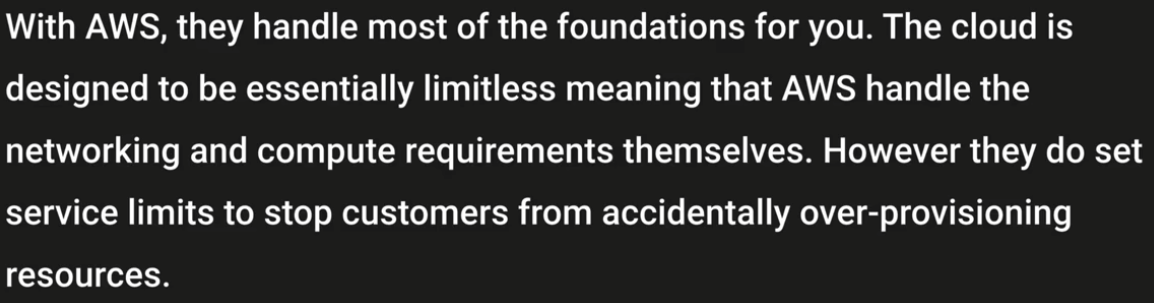
Failures ?  Recover

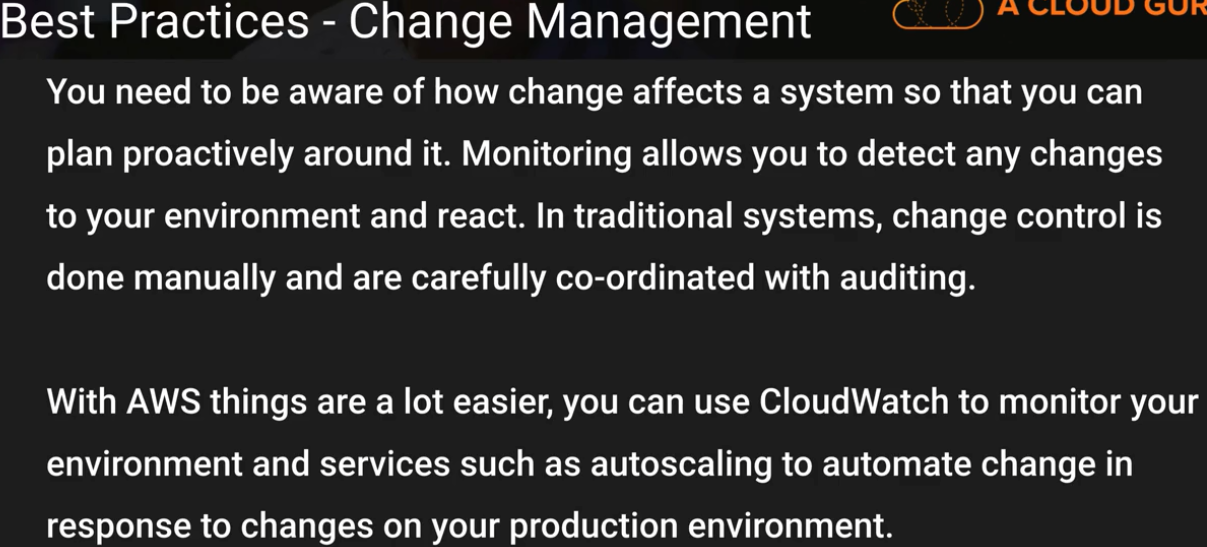
Guess capacity – problem –> end up overprovision or underprovision

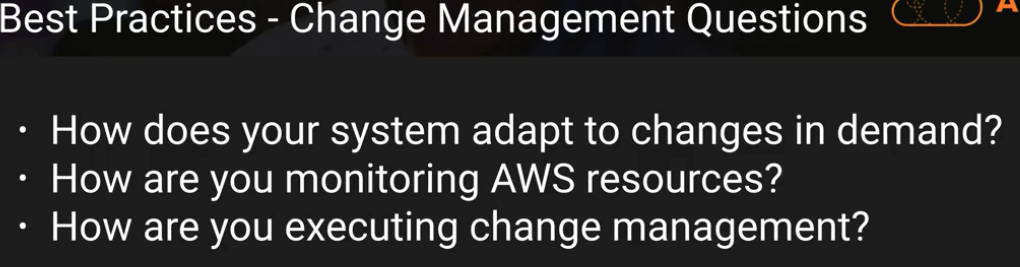


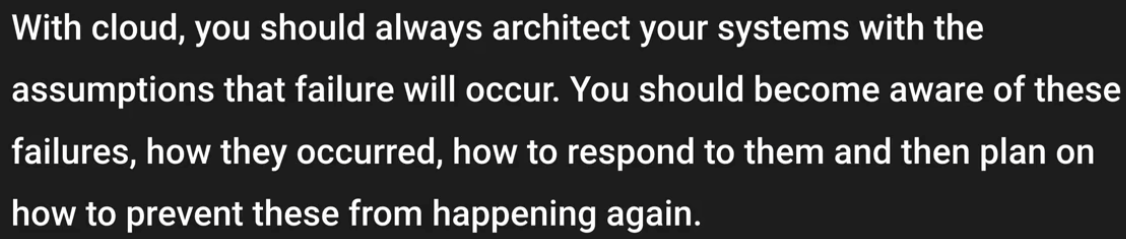


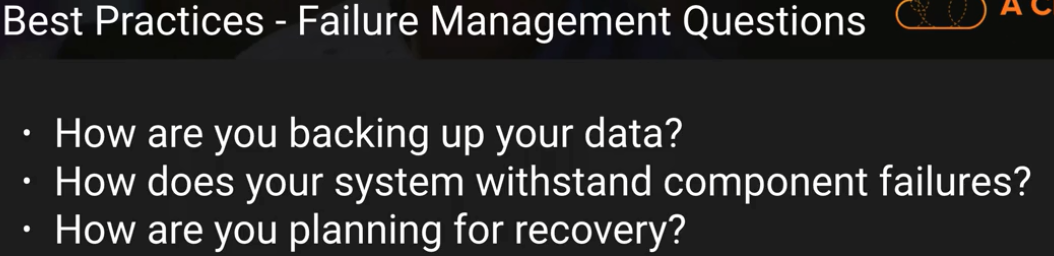


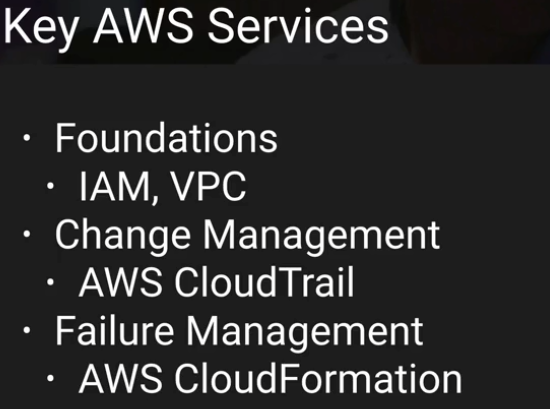












Thanks & Regards,

Shantaram Vernekar