**1. Foundations**

* Service Quotas Management: Monitor and manage service quotas to ensure resources are not exhausted, leading to failures.
* Network Planning: Design network topology to accommodate multiple environments, including public and private cloud environments.
* Resource Scaling: Ensure that your cloud environment can scale to meet demand, avoiding over-provisioning or under-provisioning.
* Backup and Redundancy: Implement sufficient backup strategies and redundancy to protect against data loss and downtime.
* Monitoring and Alerts: Establish monitoring and alert systems to track resource utilization and potential issues.
* Regional Considerations: Utilize multiple AWS regions for higher availability and disaster recovery.
* Infrastructure as Code: Use tools like AWS CloudFormation to automate infrastructure management and ensure consistency.

**2. Workload Architecture**

* Service-Oriented Architecture (SOA): Design using SOA or microservices to improve scalability and fault isolation.
* Distributed Systems: Build systems to withstand failures in one part without impacting the entire workload.
* Redundancy: Implement redundancy at all levels, including data storage, compute, and network.
* Stateless Design: Where possible, design services to be stateless to make scaling easier.
* Loose Coupling: Ensure components are loosely coupled to prevent a failure in one area from cascading.
* Data Replication: Use data replication across different availability zones to protect against data loss.
* Automated Scaling: Utilize auto-scaling features to adjust resources dynamically based on demand.

**3. Change Management**

* Automated Changes: Implement automation for changes in infrastructure to minimize human errors.
* Monitoring of Changes: Track all changes to the environment, ensuring they do not impact reliability.
* Version Control: Use version control systems for infrastructure and configuration changes.
* Continuous Integration/Continuous Deployment (CI/CD): Adopt CI/CD pipelines for consistent and reliable deployments.
* Change Audits: Regularly audit changes to ensure they follow best practices and do not introduce risks.
* Rollback Mechanisms: Design rollback strategies in case a change leads to a failure.
* Testing in Production: Where feasible, test changes in production-like environments to identify potential issues before live deployment.

**4. Failure Management**

* Automated Recovery: Implement automation for failure detection and recovery, reducing downtime.
* Fault Isolation: Design your system to contain failures within specific boundaries, minimizing their impact.
* Regular Testing: Conduct regular failure testing (e.g., chaos engineering) to validate that the system can recover from different types of failures.
* Disaster Recovery Planning: Develop and test disaster recovery plans to ensure business continuity.
* Backup Verification: Regularly verify that backups are functional and can be restored when needed.
* Incident Response Plans: Prepare and regularly update incident response plans to handle unexpected failures.
* Mean Time to Recovery (MTTR): Continuously work to reduce MTTR through automation, training, and optimization of recovery processes.