**SRE**

**Start by seeing the answer from chatgpt**

**What is Site Reliability Engineering (SRE)?**

**Site Reliability Engineering (SRE)** is a discipline that ensures that systems and services are reliable and scalable. SRE is not just about keeping things running, its about using engineering practices to ensure systems are scalable and resilient. SRE was introduced by Google to ensure their services remain reliable and scalable.

**Key Principles of SRE**

1. **Automation**: Automate repetitive tasks to reduce human error and increase efficiency. For example, instead of manually checking server health, SREs use scripts to monitor and report issues automatically.
2. **Monitoring and Observability**: Continuously monitor systems to detect and fix issues before they impact users. This involves tracking metrics like latency, traffic, errors, and saturation. For instance, if a website’s response time increases, SREs can investigate and resolve the issue quickly.
3. **Incident Response**: Have a plan for when things go wrong. This includes having runbooks (step-by-step guides) for common issues and. For example, if a server crashes, the SRE team follows a runbook to restore service and then analyzes the incident to prevent future occurrences.
4. **Service Level Objectives (SLOs)**: Define acceptable levels of service. SLOs are targets for system reliability, such as 99.9% uptime. If the system falls below this target, it triggers a response to improve reliability.
5. **Error Budgets**: Balance between releasing new features and maintaining reliability. An error budget is the maximum allowable downtime. If the system exceeds this budget, the focus shifts from new features to improving stability.

**How SRE Works:** SRE teams work closely with development and operations teams to ensure systems are reliable and scalable. Here’s a step-by-step example:

1. **Development**: Developers create a new feature for a web application.
2. **Testing**: Before deploying, the feature is tested in a staging environment to catch any bugs.
3. **Deployment**: The feature is gradually rolled out to users using a technique called canary releases, where it’s first deployed to a small subset of users to monitor its impact.
4. **Monitoring**: SREs monitor the feature’s performance using tools that track metrics like response time and error rates.
5. **Incident Management**: If an issue is detected, SREs follow incident response procedures to fix it quickly.
6. **Post-Incident Review**: After resolving the issue, the team conducts a review to understand what went wrong and how to prevent it in the future.

**Real-World Example:** Imagine an online shopping website. The SRE team ensures the site is always available and performs well, even during high traffic periods like Black Friday. They automate tasks like scaling servers up or down based on traffic, monitor the site for any slowdowns or errors, and have a plan in place for quickly resolving any issues that arise.

By applying these principles, SRE helps maintain a balance between releasing new features and ensuring the system remains reliable and performant.

(**Udemy**)

**The SRE Story**

**The Story of Your Great Idea**

**The Beginning:** Imagine you have a groundbreaking idea. You’re an expert in your field, and there’s no competition. You decide to launch your services across multiple platforms: mobile, website, and even smart TV.

**Initial Success:** After the launch, customers flock to your services. All indicators show incredible growth, and it seems your products are well-received. Customer satisfaction is high, and your app rating is a solid 4.0.

**The Downturn:** As time goes by, things start to change. The number of unhappy customers grows, and your app rating drops to 3.0. Customer dissatisfaction begins to impact your numbers. Competitors enter the market with more stable services, and you start losing customers.

**Internal Struggles:** Within your IT department, there’s tension. Developers and operations teams blame each other for the issues. Developers say the operations team can’t handle peak times and high response times. Operations blames the developers for pushing buggy code into production. Release days, once a cause for celebration, become rare and stressful.

**The Crisis:** Eventually, disaster strikes. An outage takes down all your services for half a day, damaging your company’s reputation and upsetting investors. As the founder and CEO, you’re at a loss.

**The Turning Point:** One day, an advisor suggests adopting Site Reliability Engineering (SRE). They explain that other companies have seen great results with this methodology.

**The Transformation:** Initially, both developers and operations teams are unsure about SRE, but they’re willing to try. Over the next few months, they start working together, sharing responsibilities, and understanding the importance of reliable services.

**The Results:** Your numbers improve. Customer satisfaction rises, and your app rating climbs to 4.5. Your user base grows again. New features are released more frequently without impacting customers, even during peak hours. Your company has successfully implemented SRE.

**A close-up of a document

Description automatically generated A close-up of a key terms

Description automatically generated**

**Service Level Indicators (SLIs)**

Service Level Indicators (SLIs) are metrics that measure the performance and reliability of a service. They help us understand whether a service is meeting the expectations of its users. SLIs are crucial in Site Reliability Engineering (SRE) because they provide concrete data to evaluate how well a service is performing.

**How SLIs Work in Practice:** Let’s consider a stock market website:

* **Expectation**: High security, fast response times, stability, and real-time data updates.
* **SLIs**:
  + **Uptime**: Percentage of time the website is available.
  + **Response Time**: Average time it takes for the website to respond to user requests.
  + **Error Rate**: Percentage of user requests that result in errors.
  + **Data Freshness**: Frequency of data updates.

By monitoring these SLIs, the SRE team can ensure the website meets customer expectations. If the response time increases or the error rate goes up, the team can take action to address these issues.

**Reliability: The Good, The Bad and the Ugly**

**The Good: Five-Nines and Availability**

**Five-Nines (99.999%)** is a term used to describe a very high level of availability for a service. It means the service is available 99.999% of the time, which translates to just 26 seconds of downtime in a month. However, achieving Five-Nines is extremely challenging and expensive.

**Why Not Always Aim for Five-Nines?**

While Five-Nines sounds ideal, it’s not always necessary or practical. Here’s why:

1. **Cost**: Achieving such high availability requires significant investment in infrastructure, redundancy, and maintenance.
2. **Dependency**: Services often depend on other services, which might not have the same level of availability.
3. **Business Needs**: Not all services need such high availability. For example, a service that can afford some downtime during weekends might not need Five-Nines.

**The Bad: Service Level Agreements (SLAs)**

**Service Level Agreements (SLAs)** are formal commitments between a service provider and its customers. They define the minimum acceptable level of service. If the provider fails to meet the SLA, they must compensate the customer.

For example, if a delivery service promises to deliver packages within two days and fails to do so, they might offer a refund or discount as compensation. SLAs set the baseline for reliability but should not be the ultimate goal.

**The Ugly: Service Level Objectives (SLOs)**

**Service Level Objectives (SLOs)** are the desired targets for reliability and performance. They are more ambitious than SLAs and are set by the service provider to ensure a high level of service. SLOs should be:

1. **Realistic**: Achievable within the given constraints and budget.
2. **Ambitious**: More challenging than SLAs but still attainable.
3. **Balanced**: Allowing room for innovation and new features.

**Example of SLOs in Practice:** Imagine a popular mobile app with a 4.5 rating. This app is stable, fast, and frequently updated with new features. Here’s how SLOs might be defined for this app:

* **Uptime**: 99.9% (Three-Nines), allowing for some downtime but ensuring high availability.
* **Response Time**: 95% of requests should be processed within 200 milliseconds.
* **Error Rate**: Less than 0.1% of requests should result in errors.

These SLOs ensure the app is reliable while still allowing the development team to innovate and release new features.

**Error Budgets**

An Error Budget is the amount of allowable unreliability in a system. It is derived from the Service Level Objective (SLO), which is the target level of reliability.

**How Error Budgets Work**

1. **Setting the SLO**: Define the desired reliability level. For example, an SLO might be 99.9% uptime.
2. **Calculating the Error Budget**: The Error Budget is the difference between 100% and the SLO. For a 99.9% uptime SLO, the Error Budget is 0.1%, which translates to about 43 minutes of allowable downtime per month.

**Seven Principles of SRE**

**Embrace Risk: Embrace Risk** means acknowledging that 100% reliability is impossible, especially in the realm of digital services. Here’s why:

1. **Dependency on Other Services**: Digital services often rely on other services that can fail.
2. **Cost of Reliability**: Achieving high reliability is expensive. We have limited resources and must balance spending on reliability with other needs.

**Customers want new features and improvements, and to deliver these, we must take some risks.** Embracing risk means accepting that some level of failure is inevitable and focusing on delivering what customers want most.

**Service Level Objectives (SLOs): Service Level Objectives (SLOs)** are targets for reliability. To meet a Service Level Agreement (SLA), SREs and engineering teams set Service Level Objectives. They help us define what level of reliability we aim to achieve. Here’s a quick review:

SLOs should be:

* **Aggressive but Realistic**: Challenging yet achievable.
* **Multiple**: Covering different aspects like latency, error rates, etc.

The main idea is to set realistic goals that amaze customers with reliable services.

**Eliminate Toil (replacing manual work with automation): Eliminate Toil** means reducing repetitive, manual tasks that bring no long-term value. For example, doing laundry is toil because it’s repetitive and boring. In IT, toil might be manually restarting servers or handling routine support tickets. Eliminating toil means investing in automation and efficient processes, allowing teams to focus on more valuable work.

**Monitoring: Monitoring** involves collecting, processing, and analyzing data from systems to understand their health and performance. Here’s what monitoring helps with:

* **System Health**: Knowing if the system is functioning properly.
* **Root Cause Analysis**: Identifying the causes of problems.

Monitoring allows proactive behavior, enabling teams to act quickly and efficiently when issues arise.

**Automation: Automation** means using machines to perform repetitive tasks. Benefits of automation include:

* **Reduced Human Error**: Machines don’t get tired or make mistakes due to lack of attention.
* **Faster Processes**: Machines can perform tasks more quickly than humans.
* **Scalability**: Handling more work without needing more people.

Automation is essential for scaling operations as the business grows, ensuring efficiency and reliability.

**Release Engineering: Release Engineering** is about controlling the process of delivering software from development to production. It’s like an assembly line in a factory, ensuring quality at each step. Key aspects include:

* **Process Control**: Knowing all steps from conception to production.
* **Quality Gateways**: Ensuring quality at each stage to prevent issues.
* **Critical Launch Moments**: Managing the risks associated with deploying new changes.

Release engineering ensures that new features and updates are delivered safely without impacting reliability.

**Simplicity: Simplicity** means focusing on what is essential and avoiding unnecessary complexity. It’s about making systems and processes as straightforward as possible, which is crucial for reliability. **Example:** Imagine you’re designing a new feature for a mobile app. Here’s how you can apply simplicity:

* **Essential Features**: Focus on the core functionality that users need. Avoid adding extra features that complicate the user interface.
* **User Interface**: Design a clean and intuitive interface. Users should be able to navigate the app without confusion.

**The 7 SRE Practices**

**Monitoring: Monitoring** is both a principle and a practice in SRE. It involves collecting, analyzing, and processing relevant events and facts from our services. Here’s how it works:

1. **Data Collection**: Gather data from logs, applications, and other sources.
2. **Aggregation**: Combine this data into understandable collections called metrics.
3. **Dashboards**: Display these metrics on dashboards for easy visualization.
4. **Comparison**: Compare metrics with expected behavior to detect anomalies.
5. **Notification**: Set up alerts to notify when the service is not behaving normally.

**Main Message**: Monitoring is the foundation of all other practices. It’s the “mother” of all practices because everything else depends on it.

**Incident Response: Incident Response** is about being prepared for when things go wrong. Here’s how it works:

1. **Fire-Drills**: Simulate incidents to prepare for real ones.
2. **On-Call**: Have people ready to respond to incidents at any time.
3. **Notification Plan**: Define who should be notified and how.
4. **Coordination**: Plan who should be involved and their roles.
5. **Logistic Plan**: Ensure all necessary equipment and resources are available.
6. **Communication Plan**: Decide who needs to be informed about the incident.

**Main Message**: Incident Response helps reduce damage and restore normal operations as quickly as possible.

**Root Cause Analysis and Postmortem: Root Cause Analysis (RCA)** is the process of investigating what caused an incident. **Postmortem** is the documentation of this analysis. Here’s how it works:

1. **Incident Resolution**: Focus first on reducing the impact on customers.
2. **Initial Analysis**: Sometimes, the cause is obvious and can be addressed immediately.
3. **Detailed Analysis**: Perform a thorough investigation after the incident is resolved.
4. **Postmortem Document**: Create a document listing all relevant facts in a timeline order.

**Main Message**: Root Cause Analysis and Postmortems help us learn from our mistakes and continuously improve.

**Simplicity: Simplicity** means focusing on what is essential and avoiding unnecessary complexity. Here’s why it’s important:

1. **Focus on Essentials**: Prioritize features and services that provide the most value.
2. **User Experience**: Ensure systems are easy to use and understand.
3. **Cost Reduction**: Use only the necessary resources to reduce costs.

**Main Message**: Simplicity is necessary for reliability. You can’t rely on something you don’t understand.

**Testing and Releasing**

**Testing: Testing** ensures that the software meets quality standards before it goes live. Think of it like testing a car before it hits the road. No one wants to drive an untested car!

* **Purpose**: Prevent bugs and bad designs from causing incidents.
* **Types of Tests**:
  + **Unit Tests**: Test individual components of the code.
  + **Integration Tests**: Ensure different parts of the system work together.
  + **System Tests**: Test the entire system as a whole.

**Main Idea**: Testing is about prevention. It ensures that everything works as expected in a safe environment before going live.

**Releasing: Release Engineering** involves planning and managing the process of delivering software from development to production.

* **Pipeline**: The path from development to production, often involving multiple environments like development, testing, staging, and production.
* **Automation**: Automate the pipeline to make the process faster and more reliable.
* **Continuous Delivery and Deployment**: Automate the release process to deliver updates quickly and safely.

**Main Idea**: Release engineering ensures that software is delivered efficiently and safely, minimizing the impact on customers.

**Capacity Planning: Capacity Planning** is about optimizing the use of resources to ensure they are neither underutilized nor overloaded.

* **Purpose**: Avoid wasting resources and ensure reliability.
* **Frequency**: Done regularly, such as once or twice a year.
* **Scaling**:
  + **Scale Up/Down**: Increase or decrease the power of resources.
  + **Scale In/Out**: Add or remove resources based on demand.

**Examples**:

* **Black Friday**: Anticipate higher demand and plan for additional resources.
* **Learning from Mistakes**: Use past incidents to improve future capacity planning.

**Main Idea**: Capacity planning helps save money and reduce errors by ensuring resources are used efficiently.

**Development: Development in SRE** focuses on creating internal tools and automating processes to improve reliability.

* **Role of Site Reliability Engineers (SREs)**: SREs are like blacksmiths who build tools for themselves and their colleagues.
* **Time Distribution**: SREs spend 50% of their time on development tasks and 50% on operations and incident response.
* **Automation**: Automate repetitive tasks to improve efficiency and reliability.

**Main Idea**: Having a dedicated team to build and maintain internal tools ensures that processes are efficient and reliable.

**User Experience (UX): User Experience (UX)** sits at the top of the reliability pyramid. As we move up this pyramid, we increase the reliability level of our services. Each layer builds on the one below it, with monitoring at the base and UX at the top.

**The Main Idea:** The key takeaway is that **User Experience (UX)** is the ultimate goal of all SRE practices. When performing any task, always keep in mind the impact on both internal and external users. The keyword here is **“Goal”**:

* **Internal UX**: Make tools and processes simple and efficient for your team.
* **External UX**: Deliver a reliable and innovative service to your customers.

**Site Reliability Engineer (SRE)**

A **Site Reliability Engineer (SRE)** is a professional who combines skills from both software engineering and system administration to build and maintain highly reliable and scalable systems. Here’s a more detailed look:

1. **Software Engineer**: Specializes in building software, knowing all the techniques and tools needed to create good software.
2. **System Administrator (Sysadmin)**: Specializes in maintaining systems in production, managing hardware, operating systems, networks, and more.

An SRE is essentially a blend of these two roles, focusing on ensuring that systems are reliable and can scale effectively.

**Key Responsibilities of an SRE**

* **As a Software Engineer**: Application Performance Monitoring, Pipeline Automation, Coding, Consulting on Software Design
* **As a System Administrator:** Infrastructure Monitoring, Incident Response, Capacity Planning, Postmortems, Operational Procedures.