08/03/2025 - 15/03/2025

Ciclo de vida de aplicações: monitorando, atualizando e evoluindo um software

1. Concepção;
2. Arquitetura e Design;
3. Desenvolvimento;
4. Testes;
5. Implantação;
6. Monitoramento:
    a. Garantir bom desempenho;
    b. Eficiência;
    c. Segurança;
    d. Observabilidade:
        i. Métricas;
        ii. Traços distribuídos;
        iii. Logs.
7. Manutenção:
    a. Manutenção do código;
    b. Qualidade do código;
    c. Refatoração:
        i. Redução da dívida técnica.
8. Atualização e Evolução.

Testando pós-deploy

1. **Smoke Testing**:
    a. Realizados rapidamente após o deploy para verificar se as funcionalidades básicas estão funcionando corretamente.
    b. Após cada deploy, realize testes rápidos nas principais funcionalidades para garantir que a aplicação está funcionando de forma básica.
    c. Exemplo: Envio de um formulário simples e carregamento de páginas principais em uma plataforma.
2. **Testes Unitários:**
    a. Focam em pequenas partes da aplicação (como funções ou componentes) para garantir que funcionam isoladamente.
    b. Exemplo: Garantir que um método calcula corretamente descontos em promoções.
3. **Testes E2E**:

    a. Validam o comportamento completo da aplicação, simulando o fluxo real do usuário.
    b. Implemente testes end-to-end automatizados que simulem o comportamento do usuário em um ambiente de staging. Verifique se todos os fluxos estão funcionando conforme o esperado.
    c. Exemplo: Realizar o cadastro de um usuário, fazer uma compra e emitir um comprovante.
4. **Testes de Carga**:
    a. Utilize ferramentas como Apache JMeter ou Gatling para simular múltiplos acessos simultâneos e verificar se a aplicação se mantém estável e responsiva.
    b. Imagine que você tem uma loja online que vende ingressos para um show. O teste de carga seria como simular milhares de pessoas tentando comprar ingressos ao mesmo tempo para ver se o site aguenta o tranco e não cai.
5. **Testes de Penetração**:
    a. Avaliam a segurança do sistema, buscando vulnerabilidades que podem ser exploradas.
    b. Realize testes de segurança para identificar possíveis vulnerabilidades na aplicação.
    c. Exemplo: Testar se dados de um usuário podem ser acessados indevidamente por outro usuário.
6. **Automatização**:
    a. Automatize o processo de testes, implantação e rollback para garantir a eficiência e a rapidez na correção de problemas.
    b. Imagine que você tem uma fábrica de bolos. A automatização seria como ter máquinas que misturam os ingredientes, assam os bolos e os embalam, tudo sem precisar de intervenção manual constante. Isso garante que a produção seja rápida e consistente.
7. **Monitoramento Contínuo**:
    a. Implemente um sistema de monitoramento contínuo para acompanhar o desempenho da aplicação em tempo real e identificar possíveis problemas antes que eles afetem os usuários.
    b. Imagine que você está atualizando o software do seu celular, mas a nova versão está cheia de bugs. O rollback seria como ter a opção de voltar para a versão anterior que funcionava bem, para não ficar com um celular inutilizável.

Monitorando a aplicação

1. **Configure o monitoramento em um ambiente de teste:**
    a. Implemente o monitoramento em um ambiente de teste que simule o ambiente de produção. Utilize ferramentas como Prometheus, Datadog, New Relic ou CloudWatch para coletar métricas da sua aplicação.
2. **Defina alarmes:**

    a. Com base nas métricas coletadas, defina alarmes para monitorar a disponibilidade, velocidade de resposta das APIs, tempo de carregamento das páginas, latência e utilização de recursos computacionais.
3. **Simule picos de acesso e erros:**
    a. Realize testes de carga para simular picos de acesso e verifique se os alarmes são acionados corretamente. Além disso, simule erros na aplicação para verificar se os registros de erro (logs) estão sendo coletados e analisados adequadamente.
4. **Analise os dados coletados:**
    a. Examine os dados coletados pelas ferramentas de monitoramento para identificar possíveis gargalos de desempenho, erros e áreas de melhoria na sua aplicação. Utilize os registros de erro (logs) para identificar funcionalidades com mau comportamento e planejar refatorações e atualizações.
5. **Ajuste os alarmes e métricas:**
    a. Com base na análise dos dados coletados, ajuste os alarmes e métricas para garantir que eles estejam adequados às necessidades da sua aplicação. Refine os limites dos alarmes para evitar falsos positivos e garantir que você seja alertado apenas sobre problemas reais.

Escalabilidade e elasticidade

Escalabilidade:

- Ajusta os recursos da aplicação para lidar com picos de acesso.
- Pode ser vertical (adicionar capacidade a um único recurso) ou horizontal (adicionar mais recursos ao ambiente).
- Utiliza balanceadores de carga (load balancers) para distribuir o tráfego uniformemente entre os servidores e garantir a integridade dos recursos.
- Em conjunto com os auto-scaling groups, permite adicionar ou remover recursos automaticamente com base em políticas de escalabilidade.

Elasticidade:

- Aumenta os recursos em resposta ao aumento da demanda e diminui quando a demanda diminui.
- Otimiza os custos, utilizando apenas os recursos necessários.
- Essas duas características são fundamentais na computação em nuvem, permitindo que as aplicações se adaptem dinamicamente às necessidades do momento.

Analogia:

Escalabilidade:

- Escalabilidade Vertical: Pense em trocar seu forno pequeno por um forno gigante, que consegue assar muito mais pizzas de uma vez. É como turbinar um único recurso para que ele aguente mais trabalho.
- Escalabilidade Horizontal: Imagine que, em vez de um forno gigante, você compra vários fornos menores e distribui os pedidos entre eles. Assim, você aumenta a capacidade total da pizzaria sem depender de um único ponto de falha.

Balanceadores de Carga (Load Balancers):

- Seriam como os garçons da sua pizzaria, que distribuem os pedidos entre os diferentes fornos de forma equilibrada. Eles garantem que nenhum forno fique sobrecarregado, mantendo a produção eficiente e a qualidade das pizzas. Além disso, eles verificam se todos os fornos estão funcionando corretamente e redirecionam os pedidos caso algum deles apresente problemas.

Elasticidade:

- É como ter a capacidade de adicionar ou remover fornos da sua pizzaria de acordo com a demanda. Em horários de pico, você aumenta o número de fornos para atender a todos os clientes. Já em horários mais tranquilos, você diminui a quantidade de fornos em uso, economizando recursos e evitando desperdícios.

Observando os componentes da aplicação

Observabilidade:

O que é: A observabilidade é uma abordagem de monitoramento que permite analisar detalhadamente cada componente de uma aplicação distribuída (como microsserviços) para identificar falhas, lentidão ou erros no processamento de requisições.

Por que é importante: Em arquiteturas complexas, como a de microsserviços, não basta monitorar o estado geral da aplicação. É crucial entender o que acontece dentro de cada componente individualmente.

Como funciona: A observabilidade envolve a coleta e análise de dados de cada componente, como tempo de carregamento de páginas no front-end, tempo de resposta de APIs no back-end e eficiência de consultas no banco de dados.

Objetivo: Identificar problemas de forma proativa, muitas vezes antes que eles afetem a experiência do usuário, e entender a causa raiz de falhas.
Monitoramento vs. Observabilidade: Enquanto o monitoramento responde à pergunta "Está tudo funcionando?", a observabilidade busca responder "Por que não está funcionando?".

1. **Métricas de Front-end:**
    a. Como você monitoraria o tempo de carregamento das páginas?
    b. Quais seriam os limites aceitáveis para garantir uma boa experiência do usuário?
2. **Métricas de Back-end:**
    a. Como você verificaria o tempo de resposta das APIs?
    b. Quais APIs são mais críticas e precisam de monitoramento constante?
3. **Banco de Dados:**
    a. Como você avaliaria a eficiência das consultas?
    b. Quais consultas são mais propensas a causar gargalos em momentos de alta demanda?
4. **Serviços de Terceiros:**
    a. Como você monitoraria o tempo de resposta de APIs de pagamento?
    b. Quais são os indicadores de que um serviço de terceiros está causando lentidão na sua aplicação?

observabilidade e monitoramento

A observabilidade é uma parte essencial do monitoramento de uma aplicação, especialmente em sistemas complexos e distribuídos. Aplicações modernas, como aquelas baseadas em microsserviços, geram um grande volume de dados em tempo real. Nessas situações, não é suficiente apenas monitorar métricas isoladas; é necessário observar como os diferentes componentes interagem para identificar e resolver problemas rapidamente.

Podemos definir Observabilidade como a capacidade de acompanhar o estado interno de um sistema por meio de informações geradas durante sua execução. Em outras palavras, é a habilidade de entender como o sistema está se comportando em tempo real, analisando dados como logs, métricas e tracing (rastreamento de requisições). O objetivo é garantir que aspectos importantes para a experiência do usuário final sejam monitorados e compreendidos.

Monitoramento, por sua vez, é o processo de acompanhar continuamente o estado de um sistema por meio de eventos registrados, como "uso de CPU acima de 80%" ou "falha ao conectar ao banco de dados". Ele também inclui a execução de ações reativas (como alertas em caso de falhas) e proativas (como ajustes automáticos de recursos) para manter o sistema funcionando corretamente.

Métricas:

Se estamos lidando com observabilidade e monitoramento, precisamos de algum indicador para trazer os dados importantes. Uma métrica é basicamente um indicador de nível de serviço, coletado dentro de uma série temporal.

As métricas são utilizadas para a medição de performance, disponibilidade, saturação (uso dos recursos que ultrapassa o previsto), erros e qualquer informação útil para o negócio. Portanto, a métrica é uma medição em uma janela de tempo que foca em uma propriedade do sistema.

Uma métrica sempre vai possuir um objetivo específico, e para cada ponto de atenção do sistema deve existir uma métrica correspondente. Por exemplo:

- Infraestrutura: Uso de CPU (%), espaço em disco disponível (GB).
- Execução da aplicação: Tempo médio de resposta de requisições (ms), número de requisições com erro por minuto.
- Regras de negócio: Taxa de conversão em um e-commerce (%), número de novos usuários cadastrados por dia.

Essas métricas são extraídas por meio da instrumentação e ajudam a entender o comportamento do sistema em diferentes níveis.

A partir das métricas, podemos obter informações úteis tanto para a equipe de desenvolvimento, para a operação de infraestrutura e também para a área de negócios.

Pilares da Observabilidade

1. Métricas (Metrics):
    a. Indicadores numéricos que medem o desempenho da aplicação (CPU, memória, tempo de resposta, etc.).
    b. Ferramentas: Prometheus, Grafana.
2. Traços Distribuídos (Distributed Traces):
    a. Rastreamento das requisições que circulam na aplicação, desde o início até o fim.
    b. Permite identificar gargalos e pontos de latência.
    c. Ferramenta: Jaeger.
3. Logs:
    a. Registros do funcionamento da aplicação (erros, avisos, informações).
    b. Ferramenta: Graylog.

Ferramentas de Observabilidade:

- Open Source: Prometheus, Grafana, Jaeger, Graylog, ELK Stack.
- Proprietárias: New Relic, Dynatrace.
- Nativas de provedores de nuvem (Azure, AWS, Google).

Vantagens da Observabilidade:

- Reagir a problemas de forma mais rápida e eficiente.
- Prever falhas antes que elas ocorram.
- Otimizar o desempenho da aplicação.

Respondendo a falhas

Identificação de pontos críticos: Ao receber um relatório de bug, o primeiro passo é identificar os pontos de falha para monitorar e analisar a situação.

Análise de logs: Examinamos os logs para identificar gargalos no processamento de requisições.

Configuração de alertas: Configurar alertas para detectar falhas antes que afetem muitos usuários, monitorando tanto o desempenho quanto atividades suspeitas.

Monitoramento de tráfego e comportamento na rede: Detectamos e monitoramos possíveis ataques e intrusões no sistema, analisando o tráfego em tempo real com o uso de IDS (Intrusion Detection System).

Interceptação de tráfego suspeito: Interceptamos atividades suspeitas com o IPS (Intrusion Prevention System) para evitar sobrecarga na aplicação.

Documentação de falhas: Documentamos as falhas para aprender com elas e melhorar a resiliência, o desempenho e a segurança da aplicação.

Simulação de Falhas:

1. **Crie cenários de falha**: Simule situações em que componentes da sua aplicação falham (ex: lentidão no banco de dados, erros em APIs externas).
    - **Monitore os logs**: Verifique se os logs estão fornecendo informações claras sobre a causa da falha.
    - **Teste os alertas**: Configure alertas para essas falhas e verifique se eles são disparados corretamente.

2. **Análise de Logs**:

- **Examine os logs**: Analise os logs da sua aplicação em busca de erros, avisos ou comportamentos inesperados.
- **Identifique gargalos**: Use ferramentas de análise de logs para identificar gargalos de desempenho.

3. **Implementação de Alertas**:
    - **Defina métricas**: Escolha métricas importantes para sua aplicação (ex: tempo de resposta, taxa de erros, uso de recursos).
    - **Configure alertas**: Configure alertas para quando essas métricas ultrapassarem os limites aceitáveis.
    - **Teste os alertas**: Simule situações que disparem os alertas e verifique se eles são enviados para os canais corretos (ex: e-mail, Slack).

4. **Monitoramento de Tráfego**:
    - **Use ferramentas de monitoramento de rede**: Utilize ferramentas como Wireshark ou tcpdump para capturar e analisar o tráfego de rede da sua aplicação.
    - **Procure por padrões suspeitos**: Analise o tráfego em busca de padrões suspeitos, como picos de tráfego, conexões de IPs desconhecidos ou tentativas de acesso não autorizadas.

5. **Implementação de IDS/IPS**:
    - **Configure um IDS/IPS**: Implemente um sistema de detecção e prevenção de intrusões (IDS/IPS) para monitorar o tráfego de rede da sua aplicação.
    - **Simule ataques**: Simule ataques comuns (ex: SQL injection, cross-site scripting) para verificar se o IDS/IPS detecta e bloqueia as tentativas de invasão.

6. **Documentação de Incidentes**:
    - **Crie um processo de documentação**: Estabeleça um processo para documentar todas as falhas e incidentes que ocorrerem na sua aplicação.
    - **Analise as causas**: Investigue as causas das falhas e documente as ações tomadas para resolvê-las.
    - **Compartilhe o conhecimento**: Compartilhe as lições aprendidas com a equipe para evitar que os mesmos erros se repitam no futuro.

Pilares da segurança da informação

A segurança de informação compreende um conjunto de ações e estratégias para proteger sistemas, programas, equipamentos e redes de invasões. Então, de forma geral, o intuito central das ações de segurança de informação é proteger dados valiosos de possíveis violações ou ataques.

Nesse contexto, a segurança da informação atua a partir de três pilares principais: confidencialidade, integridade e disponibilidade — que você pode conhecer pela sigla CID.

Confidencialidade

O primeiro princípio fundamental da segurança da informação é a confidencialidade, que envolve medidas para garantir que as informações sejam mantidas sob sigilo.

Um exemplo comum de ação de confidencialidade é o uso de criptografia de dados, usada em aplicativos de mensagens.

Ainda, a confidencialidade também compreende a implementação de restrições de acesso a informações específicas.

Disponibilidade

As informações também precisam estar disponíveis para as pessoas, no momento certo em que precisarem, seguindo as regras de confidencialidade.

Ou seja, as pessoas usuárias das informações devem conseguir acessá-las de todos os sistemas possíveis.

Integridade

Além de confidenciais, é indispensável que as informações permaneçam íntegras. Ou seja, que não sofram alterações não autorizadas ao longo de todo processo — que inclui tráfego, armazenamento e processamento.

Então, para garantir a integridade, são necessárias ações que previnam qualquer tipo de modificação não autorizada.

Autenticidade

Também é fundamental que as informações sejam verdadeiras e seguras. Isto é, elas precisam ser decorrentes de fontes confiáveis.

O pilar da autenticidade estabelece que é necessário registrar as pessoas autoras das informações, para que seja possível rastrear e atestar sua veracidade.

Irretratabilidade

O princípio da irretratabilidade (ou do não repúdio) estabelece que as pessoas usuárias não podem negar a autoria das informações, como uma forma de garantir a sua autenticidade.

Isso significa que, por exemplo, nem a pessoa autora e nem a receptora podem contestar qualquer transação de dados.

Conformidade

Em último lugar, a segurança da informação deve garantir que todos os processos obedeçam às leis e às normas regulamentares.

Para isso, as empresas devem desenvolver protocolos em conformidade com essas normas e, mais que isso, promover mecanismos de fiscalização para assegurar o cumprimento dos protocolos.

Identificando vulnerabilidades com OWASP

Para nos ajudar a identificar problemas comuns de segurança em aplicações, temos, por exemplo, um projeto muito conhecido e reconhecido na comunidade de desenvolvimento chamado OWASP. O OWASP é um acrônimo para Open Web Application Security Project, Projeto de Segurança de Aplicações Web, que nos fornece elementos como diretrizes e ferramentas de segurança.

Uma das diretrizes mais famosas é o OWASP Top 10, que identifica sempre as 10 principais vulnerabilidades para aplicações na web. Essa lista é constantemente atualizada, então, se consultarmos o site do OWASP, teremos acesso a várias informações sobre segurança. Além disso, o OWASP também oferece uma ferramenta para identificação de vulnerabilidades em uma aplicação, chamada Zed Attack Proxy, ou ZAP.

Identificando vulnerabilidades com Pentests

Para identificar uma vulnerabilidade utilizando o teste mencionado anteriormente utilizamos os testes de penetração, Pentests, nos quais verificamos se há algum tipo de brecha na aplicação que permita um ataque, uma interceptação de alguma mensagem, ou mesmo o sequestro de algum tipo de dado que está armazenado.

Realizamos esse processo de identificação de vulnerabilidade com os pentests por etapas. Primeiro, coletamos informações da aplicação, entendemos seu funcionamento interno, seus principais mecanismos de segurança, e fazemos um planejamento e escopo. Nessa parte de planejamento e escopo, também analisamos e consultamos as principais vulnerabilidades identificadas pela lista da OWASP.

A partir disso, identificamos alguns pontos de melhoria a partir da execução desses testes de penetração e exploramos aplicações em relação à segurança, para gerar um relatório de

vulnerabilidades. Esse relatório aponta os pontos fracos em termos de segurança na nossa aplicação, para que possamos mitigá-los antes mesmo que essa aplicação tenha seu release em ambiente de produção.

Depois de mitigar esses pontos de vulnerabilidade, realizamos novamente os testes de penetração para verificar se, de fato, esses problemas foram corrigidos. O objetivo é garantir que a aplicação seja o mais inviolável possível, para que ninguém consiga atacar, interceptar e roubar qualquer tipo de dado ou informação.

Modelos de boxes

Para isso, podemos usar um modelo como o de boxes: White Box, Gray Box e Black Box.

White box

No White Box, fazemos um teste de um ataque mais profundo. Isso, porque conduzimos um ataque com base nas informações do sistema. Conhecemos o sistema em profundidade, então coletamos todas essas informações e conduzimos esses testes para verificar o que conseguimos extrair daquela aplicação em termos de dados e o que conseguimos explorar em termos de segurança. Geralmente, usamos essa abordagem quando estamos na fase de teste de uma aplicação.

White box

No Black Box, fazemos a simulação de um ataque real, que, em geral, as pessoas por trás daquele ataque não sabem ou sabem pouquíssimas informações sobre a estrutura interna da aplicação e seus mecanismos de segurança. Com isso, testamos as medidas de proteção e de resposta aos ataques.

Gray box

Também podemos fazer um teste de penetração que incorpora algumas ideias do White Box e do Black Box, fazendo essa combinação. Existe uma troca de informações quando concebemos esse teste de penetração no modelo do Gray Box. Usamos esse modelo frequentemente para aplicações web, pois, às vezes, temos mais informações sobre a aplicação disponíveis na web, e assim esses ataques podem ser melhor preparados.

Ferramentas para Pentest

Há uma grande variedade de testes de penetração que podemos conduzir em uma aplicação. Eles serão aplicados de acordo com a especificidade da aplicação. Podemos fazer, por exemplo, testes de upload de arquivos com extensões desconhecidas ou estranhas, para ver

se a aplicação aceita. Se aceitar, identificamos uma vulnerabilidade. Podemos tentar fazer o login utilizando cookies de outro usuário para ver como a aplicação reage. Para isso, podemos utilizar diferentes ferramentas, como o ZAP, que está dentro do projeto OWASP, e também o Burp Suite, o Metasploit, o Nikto e o Nmap. Cada uma dessas ferramentas possui especificidades de uso, então devemos selecionar e utilizar de acordo com o tipo de pentest que queremos conduzir em uma aplicação.

Explorando ferramentas de IaC

A automação de infraestrutura é essencial em ambientes modernos, onde a agilidade e a consistência são indispensáveis. Nesse contexto, o Infrastructure as Code (IaC) surge como uma abordagem revolucionária, permitindo que a infraestrutura seja gerenciada de forma programável, assim como o código de uma aplicação.

Com o IaC, você pode criar, configurar e manter ambientes de forma consistente, evitando erros manuais e reduzindo o tempo de provisionamento. Além disso, ele é uma base sólida para práticas como DevOps e gerenciamento de infraestrutura em ambientes de nuvem.

Entre as ferramentas mais populares no universo do IaC, destacam-se o Ansible e o Terraform.

Ansible

O Ansible é uma ferramenta de automação de TI que simplifica tarefas como gerenciamento de configuração, orquestração de serviços e implantação de aplicativos. Sua abordagem baseada em YAML e sua operação sem a necessidade de agentes (agentless) tornam o Ansible acessível e eficiente, mesmo para iniciantes.

Com o Ansible, você pode:

- Automatizar tarefas repetitivas e complexas.
- Garantir consistência nas configurações entre ambientes.
- Gerenciar servidores locais e em nuvem de maneira integrada.

Terraform

O Terraform, por sua vez, é focado em provisionamento de infraestrutura. Ele permite a definição e o gerenciamento de recursos em múltiplos provedores de nuvem, como AWS, Azure e Google Cloud, usando uma linguagem declarativa.

O Terraform é ideal para:

- Criar e destruir infraestrutura de forma programada.
- Gerenciar a infraestrutura como código, facilitando revisões e versionamento.
- Garantir a portabilidade e integração com diversos provedores.