

Localize

Teste Arquiteto de Software

Teste de Arquitetura de Software da [Localize lawtech](#).

REQUISITOS

Por gentileza, observe os quesitos abaixo.

Projeto 1:

- Stack: .Net 4
- Banco de Dados: MSSQL
- Gera dados de serviços prestados
- Possui consumo constante de informações de APIs externas
- Recebe pedidos de serviços do Projeto 2
- Precisa de manutenção casualmente, ainda recebe atualizações

Projeto 2:

- Stack: .Net 7
- Banco de Dados: PostgreSQL
- Consome dados enviados pelo Projeto 1 para gerar insights e relatórios
- Precisa sincronizar dados do Projeto 1 de forma constante
- Há como solicitar sincronização manual com o **Projeto 2**
- Recebe atualizações constantemente

Ambos projetos utilizam a nuvem AWS e seus serviços.

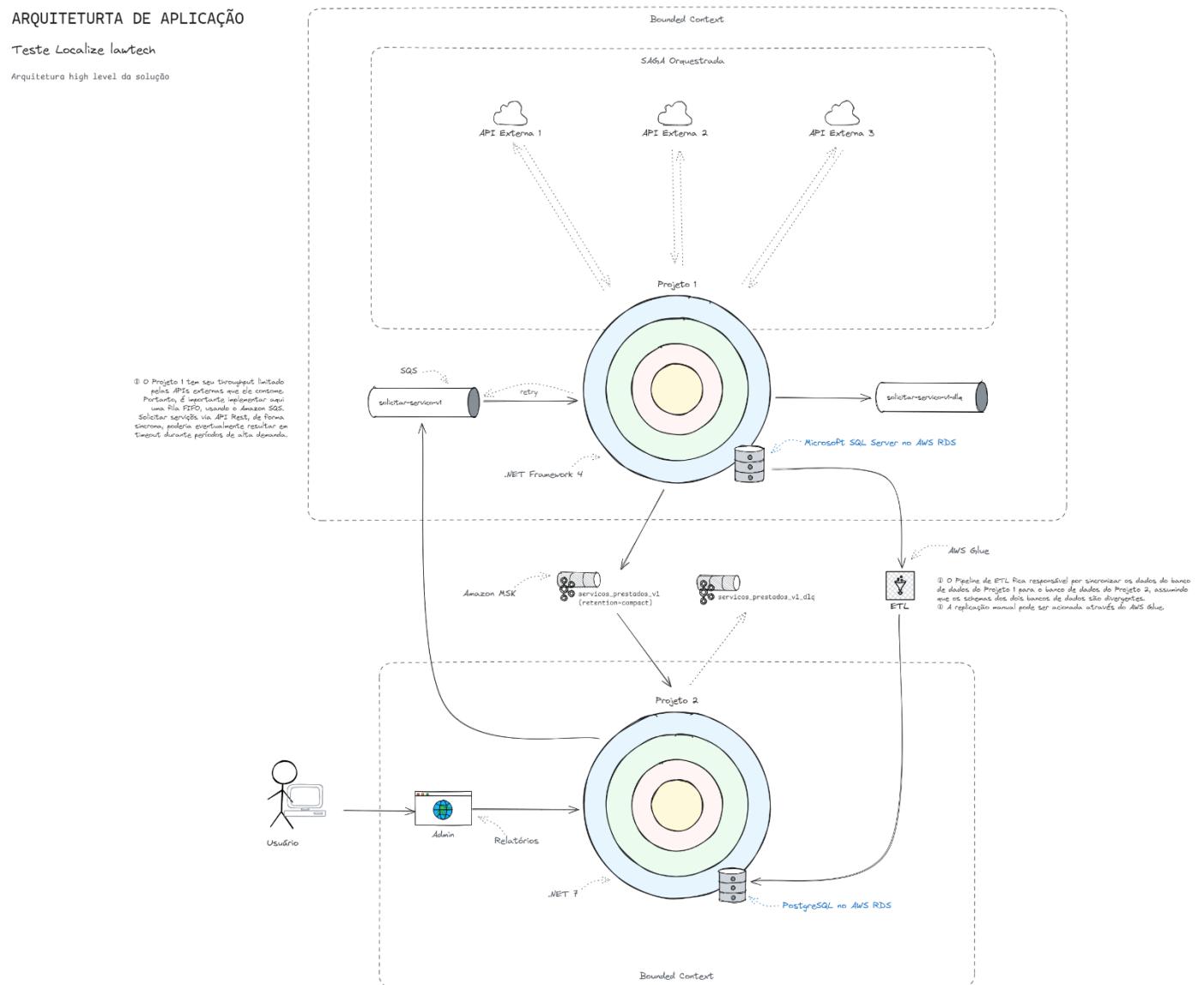
Com estes dados, responda às seguintes questões:

1. Faça um diagrama de sistema para uma forma de comunicação entre ambos projetos.
2. Quais ferramentas de observabilidade você usaria para avaliar a continuidade da comunicação entre os projetos?
3. Qual pipeline você usaria para o Projeto 2 consumir dados do Projeto 1? Justifique.
4. Qual prática de DevOps você aplicaria ao Projeto 1, baseado nas informações atuais?
5. Sobre DevSecOps, quais práticas você aplicaria para sua solução? Explique sua resposta.

Documentação

Arquitetura de Aplicação

Arquitetura high-level da solução.



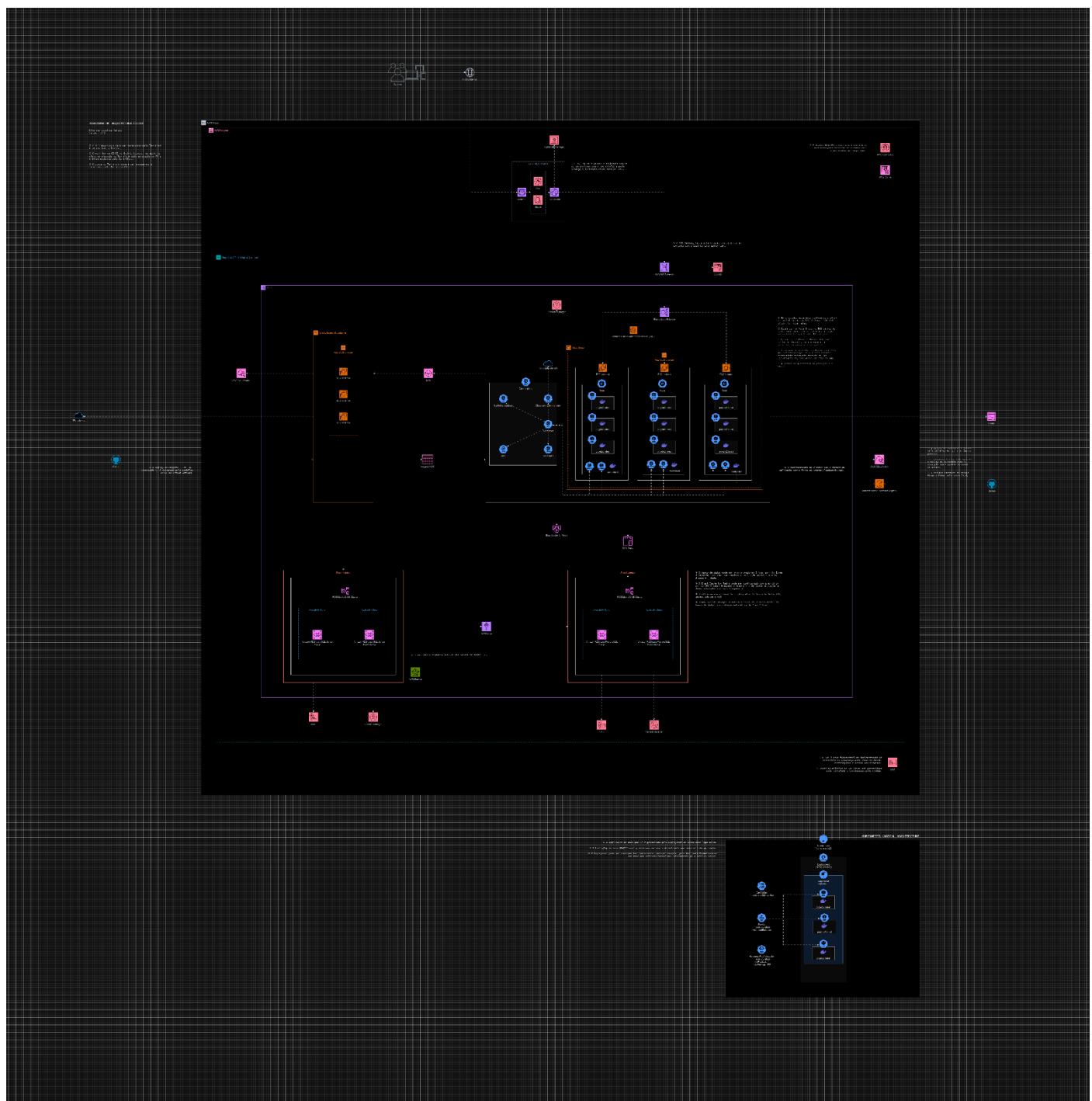
*A imagem pode ser visualizada em alta definição no GitHub em
<https://github.com/dannevesdantas/teste-localize/blob/main/docs/arquitetura-de-aplicacao/arquitetura-de-aplicacao.png>*

Arquitetura Cloud

Cloud provider: AWS

[Clique aqui para ver no draw.io ↗](#)

Mais detalhes sobre a motivação para adoção de cada serviço de nuvem estão disponíveis no diagrama.



*A imagem pode ser visualizada em alta definição no GitHub em
<https://github.com/dannevesdantas/teste-localize/blob/main/docs/arquitetura-cloud/arquitetura-cloud.drawio.png>*

Questões

2. Quais ferramentas de observabilidade você usaria para avaliar a continuidade da comunicação entre os projetos?

Quanto a observabilidade, há várias opções disponíveis. Algumas delas são, por exemplo:

- [Datadog](#)
- [New Relic](#)
- [Dynatrace](#)
- [ELK Stack](#)
- [Jaeger](#)
- [Prometheus](#)
- [Grafana](#)
- [Amazon CloudWatch logs](#)
- [OpenTelemetry](#)
- [Istio](#) Service Mesh + [Kiali](#)
- Entre outros

No diagrama de arquitetura cloud, optei por utilizar o **CloudWatch logs** da AWS* pelo fato deste fazer parte do conjunto de serviços gerenciados da AWS, eliminando assim a princípio a necessidade de adotar ou contratar uma plataforma externa.

Para avaliar a continuidade da comunicação entre os projetos é importante também implementar tracing distribuído, através de correlation IDs. Um Service Mesh como o [Istio](#) por exemplo pode prover essa funcionalidade.

**É importante destacar que essa pode não ser a melhor opção em todos os cenários e pode eventualmente também não ser economicamente viável caso haja eventuais limitações de orçamento.*

3. Qual pipeline você usaria para o Projeto 2 consumir dados do Projeto 1? Justifique.

Quanto a replicação de dados do Projeto 1 para o Projeto 2, algumas opções amplamente adotadas pela indústria são:

Síncronas:

- REST APIs
- Batch Processing
- Pipeline de ETL (como o SSIS da Microsoft por exemplo)
- Entre outros

Assíncronas:

- Streaming de eventos (como o Apache Kafka por exemplo)
- Message broker (como o RabbitMQ por exemplo)
- Change data capture (CDC)
- Arquitetura Data Mesh
- Entre outros

Cada solução possui seus prós e contras. A solução ideal para cada cenário **pode variar muito de acordo com o contexto**. Uma estratégia comum que vem sendo adotada em larga escala no mercado é a combinação do [Debezium](#) com o Apache Kafka, para integrações em Near real-time (NRT). Porém, essa pode não ser a melhor solução em todos os cenários.

⚠️ *Para tomar essa decisão de forma mais assertiva seria necessário entender mais profundamente alguns aspectos importantes como por exemplo o nível de consistência necessário, o contexto de negócio e a volumetria envolvida na replicação de dados do Projeto 1 para o Projeto 2 para uma tomada de decisão mais afetiva a respeito da replicação.*

No diagrama de arquitetura cloud, optei por implementar um pipeline de ETL usando o **AWS Glue*** pelo fato deste fazer parte do conjunto de serviços gerenciados da AWS. O AWS Glue é um serviço serverless que permite aplicar transformações nos dados através de uma interface simples e intuitiva, o que facilita bastante a migração e replicação de dados quando os schemas de origem e de destino são diferentes.

Outra opção interessante seria também a implementação da replicação fazendo streaming de eventos, incluindo no corpo dos eventos o estado dos agregados e entidades. Porém, essa opção pode resultar em complexidades adicionais caso os schemas de origem e destino sejam muito divergentes, exigindo transformações de dados adicionais.

*É importante destacar que essa pode não ser a melhor opção em todos os cenários e pode eventualmente também não ser economicamente viável caso haja eventuais limitações de orçamento.

4. Qual prática de DevOps você aplicaria ao Projeto 1, baseado nas informações atuais?

O Projeto 1 se trata de uma aplicação .NET Framework 4. Sendo assim, pode ser executada somente em servidores Windows. Essa limitação impõe algumas restrições.

Para desenvolver a cultura DevOps, podemos implementar algumas práticas como:

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Feedback Contínuo
- Shift Left
- Entre outros

Para a implementação de pipelines de CI/CD existem algumas ferramentas que são amplamente adotadas pelo mercado, como por exemplo:

- GitHub Actions
- GitLab
- BitBucket
- Azure DevOps
- Jenkins
- Spinnaker da Netflix
- XL Release
- Entre outras

É importante destacar que DevOps se trata de uma cultura, e não somente de um conjunto de ferramentas.

Devido a popularidade, simplicidade e versatilidade eu pessoalmente optaria pelo **GitHub Actions*** para implementação de pipelines de CI/CD, em conjunto com ferramentas de Análise Estática de Código Fonte (SAST), como o [SonarCloud](#) por exemplo em conjunto também com alguma ferramenta de Teste Dinâmico de Segurança de Aplicações (DAST), como o [ZED Attack Proxy](#) (open source) por exemplo.

Sob a ótica de DevOps, é importante também optar por algum Branching Model, como o [GitFlow](#) por exemplo, a ser definido em conjunto com a equipe de desenvolvimento.

Continuous Delivery (CD)

Os containers revolucionaram a forma como fazemos deploy das aplicações. Apesar de ser tecnicamente possível [executar containers Windows na AWS](#), o suporte para containers Windows possui algumas limitações importantes que precisam ser levadas em consideração. Devido as limitações dos containers Windows, eu descartaria essa opção.

Uma alternativa interessante para aplicações .NET Framework na AWS é o **Elastic Beanstalk***, um serviço PaaS gerenciado que permite configurar grupos de auto scaling com facilidade. Ao optar por um serviço PaaS o time técnico pode focar seus esforços no desenvolvimento de novas features, delegando a manutenção, o gerenciamento e a atualização da infraestrutura adjacente para o provedor de nuvem. Por esses motivos a minha primeira opção seria o Elastic Beanstalk.

Além do Elastic Beanstalk, podemos claro sempre utilizar instâncias EC2 no modelo IaaS, porém, nesse caso, a equipe deve ficar encarregada de gerenciar o sistema operacional e também aplicar atualizações.

**É importante destacar que essa pode não ser a melhor opção em todos os cenários e pode eventualmente também não ser economicamente viável caso haja eventuais limitações de orçamento.*

5. Sobre DevSecOps, quais práticas você aplicaria para sua solução? Explique sua resposta.

No desenho da solução optei por implementar cofres de senhas, utilizando o **AWS Secrets Manager**.

O AWS Secrets Manager fica encarregado de armazenar e rotacionar de tempos (*por padrão de 7 em 7 dias*) automaticamente as senhas dos usuários admin dos bancos de dados SQL Server e PostgreSQL, indo de encontro as recomendações do [Well-Architected Framework da AWS](#).

As políticas do **IAM** da AWS permitem também a aplicação do princípio de menor privilégio (PoLP) restringindo o acesso a recursos na nuvem.

A fim de contribuir com a segurança das aplicações, podemos implementar Análise Estática de Código Fonte (SAST) como o [SonarCloud](#), em conjunto também com alguma ferramenta de Teste Dinâmico de Segurança de Aplicações (DAST) como o [ZED Attack Proxy](#) (open source) por exemplo.

Ferramentas de SAST e DAST podem ser implementadas diretamente no pipeline de CI, a fim de evitar que possíveis falhas de segurança sejam integradas na branch main. Além disso, o desenvolvedor pode instalar os plug-ins do [SonarLint](#) diretamente na IDE, para que seja possível identificar possíveis falhas de segurança logo no início do processo de desenvolvimento.

Observações adicionais

- É importante destacar que a solução proposta nos diagramas de arquitetura acima se trata de uma solução genérica, baseada em requisitos genéricos que, portanto, pode não ser a solução ideal em todos os cenários e contextos.
- Ao elaborar o diagrama de arquitetura, não levei em considerações possíveis e eventuais limitações de orçamento. Eventuais limitações de orçamento podem interferir no desenho da solução.
- Notei que nos requisitos no Projeto 2 parece haver um pequeno erro de digitação (*grifado em amarelo no print abaixo*). Portanto, levei em consideração que a sincronização de dados será feita com o Projeto 1, ao invés do próprio Projeto 2.

Projeto 2:

- Stack: .Net 7
- Banco de Dados: PostgreSQL
- Consome dados enviados pelo Projeto 1 para gerar insights e relatórios
- Precisa sincronizar dados do Projeto 1 de forma constante
- Há como solicitar sincronização manual com o Projeto 2
- Recebe atualizações constantemente

Para esclarecimentos adicionais, fique à vontade para entrar em contato através do LinkedIn em <https://www.linkedin.com/in/dannevesdantas/>