Podcast

Disciplina: Práticas da cultura DevOps no desenvolvimento de sistemas

Título do tema: Selecionando Conhecendo Ferramentas do DevOps no Desenvolvimento Web.

Autoria: Anderson Pereira de Lima Jerônimo

Leitura crítica: Stella Marys Dornelas Lamounier

Olá, ouvinte! No podcast de hoje vamos falar sobre os conceitos essenciais do movimento cultural DevOps aplicada ao desenvolvimento de sistemas *Web*. As equipes DevOps precisam ficar ligadas no mercado quanto ao uso de ferramentas e, no decorrer desse podcast vamos abordar duas tecnologias que usualmente são utilizadas no desenvolvimento e infraestrutura de projetos de tecnologia da informação, são elas: GIT e Docker (*containers*). A primeira tecnologia que iremos falar, é o GIT. Essa ferramenta de controle de versão, trabalha através do modelo distribuído, neste modelo não precisa necessariamente ter servidor centralizado para gerenciar todas as operações de versionamento. A principal vantagem do GIT é o compartilhamento do projeto com os seus membros das equipes, porque todas as operações são realizadas no próprio ambiente local e só depois é compartilhado para um repositório (servidor) remoto.

Já nas ferramentas que trabalham em modo centralizado, como TortoiseSVN e Subversion, podem sobrecarregar o servidor com as requisições constantes de diferentes iterações das equipes no projeto e dificultando o compartilhamento do projeto em servidores diferentes. Devido a isto, o modelo distribuído ganhou mais popularidade entre a comunidade de desenvolvedores, e ainda podemos destacar o próprio surgimento da tecnologia GIT, que se iniciou com Linus Torvalds, criador do Sistema Operacional Linux, que tinha como desafio na época controlar as versões do desenvolvimento do kernel das distribuições do Linux. Algo importante na ferramenta de controle de versão, como GIT, é que ela permite criar o que nós chamamos de *snapshots* do projeto, ou seja, como se fosse tirar uma foto de um estágio do desenvolvimento do projeto, criando através desse *snapshot*, uma *tag* de versionamento.

Além dessa funcionalidade, podemos monitorar os locais onde ficam as versões do projeto, esses locais são conhecidos como repositório e dentro desse repositório podemos criar e gerenciar o *branche*. Um repositório pode ter apenas um *branche* principal e/ou vários *branches*, como por exemplo na maioria dos projetos, geralmente há um branche para ambiente de produção e um outro para ambiente de teste. Isso traz uma melhor leitura para monitoramento e adequações que podemos agregar decorrer do desenvolvimento do projeto sem comprometer o projeto que está em produção. Também garante aos usuários a entrega contínua de novas versões, por se tratar de um pilar importante da metodologia DevOps. Próxima tecnologia que

iremos falar agora, é o Docker. Essa ferramenta ganhou nos últimos anos uma certa notoriedade, devido a facilidade de obter um ambiente de desenvolvimento estável, com ambiente de fácil compartilhamento com outros membros da equipe DevOps. Existe até uma frase famosa entre os desenvolvedores que relatam a dificuldade de compartilhar o ambiente desenvolvimento, que diz assim: “O sistema não roda na minha máquina”, isso se da por conta do projeto trabalhar com diferentes libs ou frameworks, que alguns casos podem funcionar bem em um sistema operacional especifico e em outro não. Uma alternativa para este problema antes do uso Docker, foi o uso de máquinas virtuais, utilizando os famosos programas como VMWare e VirtualBox, mas esse tipo de solução não consegue ser tão rápida nas respostas, porque depende da camada do *hypervisor* para acessar os recursos da máquina hospedeira e, dependendo da configuração do hardware pode acarretar uma certa lentidão e queda na *performace* do ambiente de desenvolvimento. Com isso, essa solução com máquinas virtuais não é indicada quando você precisar ter mais de um servidor na mesma máquina hospedeira, entrando em cena nessas situações o Docker, que diferente das máquinas virtuais, não precisa ter a camada do hypervisor para poder se comunicar diretamente com recursos do sistema operacional, ou seja, ele consegue se comunicar de forma direta ao sistema operacional, que faz aumentar consideravelmente a *performace* e fluidez na comunicação com as aplicações. Essa facilidade na comunicação é indiferente entre sistemas operacionais, permitindo o compartilhamento de forma fácil sem perder as principais funcionalidades.

Nesse modelo de ambientação com Docker, podemos gerenciar melhor os microserviços que aplicações utilizam para funcionar. Embora o Docker seja ideal para gerenciar containers únicos, quando você começa a usar mais containers, o gerenciamento, que nós chamamos de orquestração, pode ficar mais complicada. Então, entra em ação o Docker Compose que é um orquestrador de containers da Docker. E como funciona um orquestrador em uma orquestra? Ele rege como uma banda deve se comportar/tocar durante uma determinada apresentação ou música. Com o Docker Compose é a mesma coisa, mas os maestros somos nós! Nós que iremos reger esse comportamento através do arquivo chamado docker-compose, semelhante ao Dockerfile, escrito em YAML (acrônimo recursivo para *YAML Ain’t Markup Language*) é um formato de codificação de dados legíveis por humanos, o que torna fácil de ler e entender o que um Compose faz! Um exemplo prático de como funciona o Docker Compose é: imagine que temos uma aplicação Javascript ou Ruby e que essa aplicação depende de um banco de dados MySQL e, para disponibilizar essa aplicação na internet, queremos utilizar um proxy na frente. O cenário é bem típico e uma pessoa de infraestrutura configura esse ambiente fácil em menos de um dia. Agora imagina que para cada cliente, precisamos realizar esse setup pelo menos umas três vezes: um para Desenvolvimento, Homologação e outro para Produção. Complicou um pouco, né? Outro detalhe, se esses ambientes estão em cloud e existe uma Instância/VM para cada aplicação, isso pode gerar muito custo, recurso

desperdiçado e tempo desnecessário gasto, sem contar o trabalho que dá para montar essa infra toda vez que surge um projeto novo. E no mundo de hoje, precisamos ser ágeis para entregar valor ao cliente.

Outra grande desvantagem desse processo é que ele é altamente manual, logo existe uma possibilidade de acontecer um erro humano. Bom, falei isso tudo para entender qual a “dor” que tínhamos na infra e explicar como o Docker Compose pode ajudar nessa árdua tarefa! O Compose File, é um arquivo Compose, que conseguimos descrever a infraestrutura como código e como ela vai se comportar ao ser iniciado. Se digo que preciso de um banco de dados para rodar minha aplicação Javascript/Ruby, descrevo isso no meu Compose e digo que minha aplicação depende do meu container de banco de dados MySQL para rodar. Outra coisa legal, é que podemos definir o comportamento que o Docker vai ter caso um dos containers venha a falhar. Descrevo no Compose que, caso o banco de dados falhe por algum motivo, o Docker que suba outro imediatamente. Consigo isolar essas aplicações em uma rede separada e quais volumes essas aplicações vão utilizar para persistir os dados. Vamos subir todos esses serviços descritos no Compose com apenas um comando. Essa automatização é muito bacana, a forma que temos de orquestrar os serviços de forma declarativa através do YAML.

Este foi nosso podcast de hoje! Até a próxima!