Разбор реального кейса: MVP инфраструктурной платформы

Не забудь включить запись!

План

- Постановка задачи
- BSA - как мы организовали инфраструктуру
- Base Layer - GCP
- Base Layer - Yandex Cloud
- Service Layer - GCP
- QA

Постановка задачи

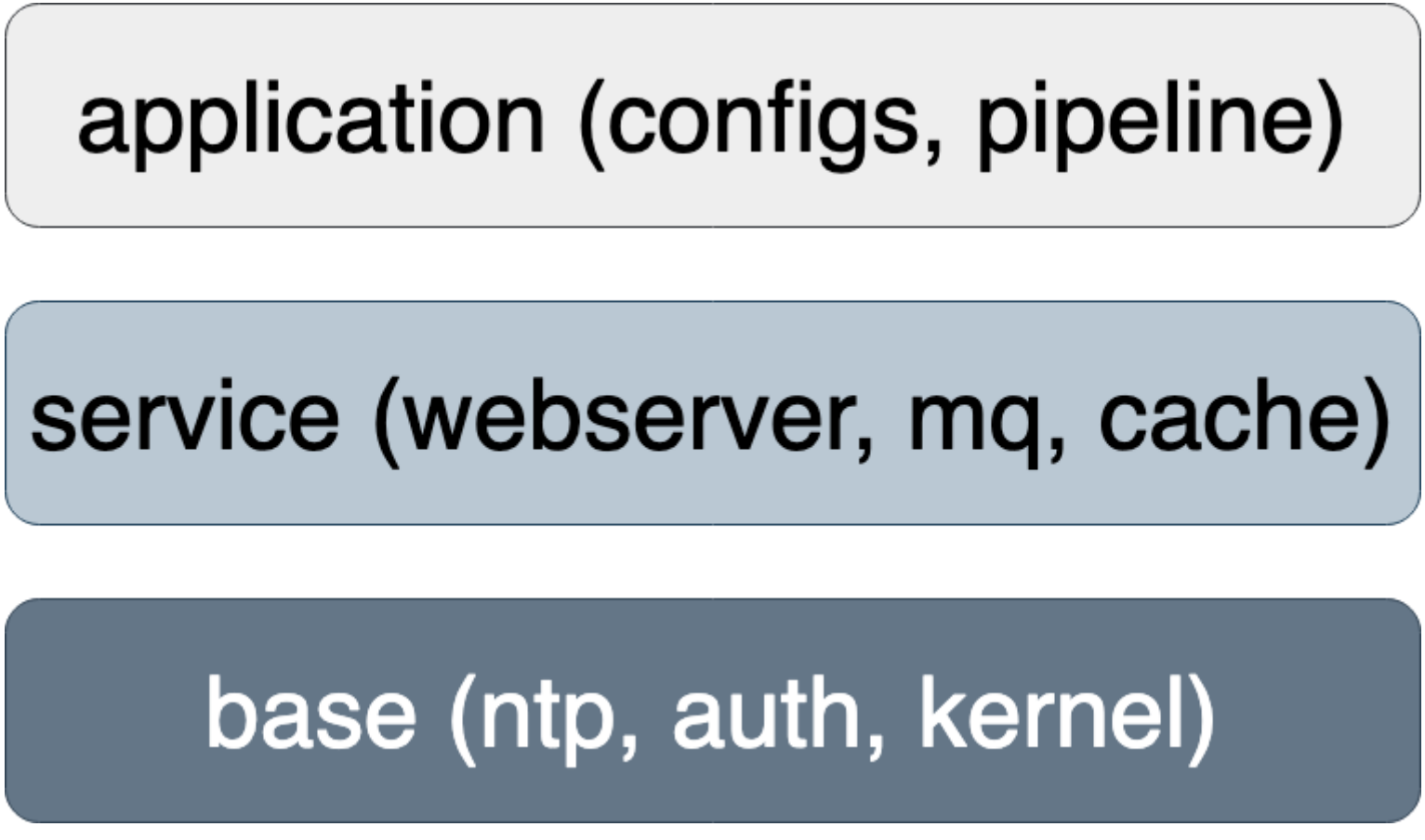
- В наличии несколько продуктов, условно готовых к запуску в Kubernetes
- Один или несколько кластеров на продукт
- Надежная и отказоустойчивая инфраструктура
- По возможности managed Kubernetes (но определенным продуктам нужен Yandex Cloud)
- Тиражируемость решения на другие продукты/команды

Постановка задачи

- Предоставить продуктовым командам кластера Kubernetes
- Один или несколько кластеров на продукт
- Подходы IaC и CI/CD для управления кластерами
- Два сценария:
  - Managed Kubernetes в GCP
  - Self-hosted Kubernetes в Yandex Cloud
- Тиражируемость решения на другие продукты/команды

Base Service App (BSA) модель

При использовании подхода IaC модель BSA позволяет менять и переиспользовать сущности в слоях, не влияя на другие слои.



Base Service App (BSA) модель

- Разный **Base layer** для разных облаков (GCP + Yandex Cloud)
- Максимально идентичный **Service layer**, способный разворачиваться на любом окружении
- При развертывании **App layer** предполагается, что Base и Service уже развернуты
- Разные окружения создаем из одного инфраструктурного кода

Используемые инструменты

GitLab CI

GCP:

- Base - Terraform
- Service - helmfile
- App - Helm

Yandex Cloud:

- Base - Terraform + ansible + yc + sops + kubespray
- Service - helmfile
- App - Helm

Base | GCP

Base | GCP

Два полностью* независимых окружения на продукт внутри одного проекта - **develop** и **production**. Каждое окружение включает в себя:

- GKE
- Cloud SQL
- Cloud Network Services (DNS + VPC + VPN + etc...)
- etc...

Особенности

- Создаем приватные региональные кластера с включенным автомасштабированием
- Используем последнюю доступную версию Kubernetes в регионе (и обновляемся на новую в полуавтоматическом режиме)
- Необходим доступ снаружи к Cloud SQL
- Используем VPC Native
- Отключаем Stackdriver
- GCR как Container Registry

С какими нюансами столкнулись

- Для приватных кластеров необходимо добавлять Cloud Nat если необходим доступ с worker нод наружу
- Cloud VPN не работает (у нас) если использовать стандартный сетевой драйвер для Kubernetes
- IP адрес для Cloud SQL instance не добавить в предсказуемый subnet range (необходимо для VPN)
- При обновлении кластера в Terraform необходимо увеличивать timeout's

С какими нюансами столкнулись

- В приватных кластерах доступ с master нод на worker ноды ограничен, для некоторых сервисов нужны дополнительные правила firewall (**Prometheus operator**, **cert-manager**)
- Для регионального кластера нужны StorageClass с региональными дисками (стоимость существенно выше)
- Если размер instance для Cloud SQL больше (autoscaling), чем был в Terraform - instance будет пересоздан
- Нельзя удалить Cloud SQL instance и создать новый с таким же именем

GKE

Два GKE кластера на продукт:

- Кластер для Develop окружения и Feature веток
- Кластер для Staging и Production окружений

Преследуемые цели:

- Максимальное разнесение Develop и Production окружений
- "Тренировочный" Develop кластер

Terraform state окружений храним в двух storage buckets

Cloud SQL

- Два автомасштабируемых* HA CloudSQL инстанса баз данных, распределенных в пределах региона - **Develop** и **Production**
- Read-only реплика для Production окружения
- Автоматическое присоединение инстансов к VPC
- Ограничение доступа по публичным IP адресам (выделенная AS)
- Создание Bastion для подключения из других мест
- Приватные DNS зоны для подключения изнутри VPC

Base | Yandex Cloud

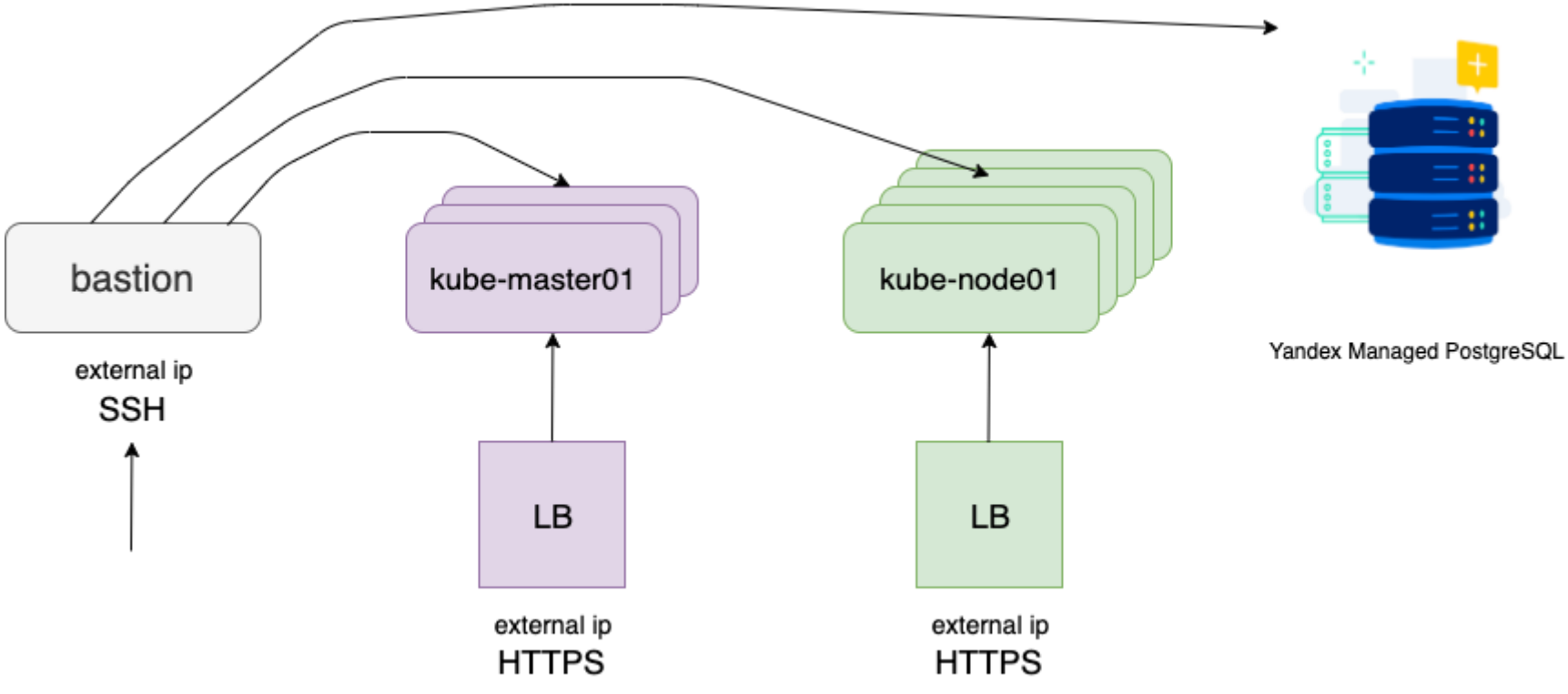
Концепция

- Два статичных окружения **develop** и **stage**
- Произвольное количество **production** окружений
  - production-russia01, production-russia02, production-latvia01
- Managed Kubernetes не подходит требованиям продукта
- Но можно Managed PostgreSQL 🥳
- Поддержка параметризации окружений и настроек кластеров (количество nodes, размер vm, секреты)

Особенности

- Подразумеваем, что управляем всем Каталогом в рамках облака
- Распределяем виртуальные машины по зонам доступности
- Дополнительные диски для нужд эксплуатации кластера
- Виртуальные машины кластера не имеют публичных IP-адресов
- Пользовтаельский трафик в кластер попадает через L4 LB
- Управляющее взаимодействие с кластером через другой L4 LB
- Управляющее взаимодействие с серверами через bastion-хост

То же самое, но схемой



С какими нюансами столкнулись

- Terraform provider поддерживает не все сервисы облака
- Виртуальные машины необходимо преднастроить
- Для виртуальных машин без внешнего IP нужен NAT
- Выбор способа развертывания Kubernetes
- Выбор реализаций для сети и хранилища в кластере
- Нетривиальный пайплайн

Пайплайн

- Запускается кнопкой и при указании целевого окружения
- Создание, масштабирования и перенастройки кластера

Этапы выполнения

- Выполнение проверок
- Расшифровка секретов
- Terraform plan/manual apply
- Базовая настройка виртуалок с помощью Ansible
- Запуск kubespray для установки/настройки кластера
- Применение базовых манифестов (ролевая модель, аккаунты)

Service | GCP

Helmfile

- Позволяет установить несколько сервисов на окружение с минимальной кастомизацией инфраструктурного кода
- Умеет развертывать Helm Charts по порядку
- Diff перед установкой
- Ждем релиза (**v1**)
- Смотрим на другие решения (**Terraform Helm Provider**, **Pulumi**, etc...)

Особенности

- Некоторые чарты плохо работают с CRD (**cert-manager**, **prometheus-operator**). Поэтому требуется дополнительная логика установки и удаления CRD в pipeline
- Нельзя установить чарты с `serviceMonitor` (**nginx**, **prometheus-exporters**, **loki**, etc...), если нет CRD от prometheus-operator
- Выкатываем все как helm charts, в том числе используем собственный репозиторий (**oauth2-proxy**, **prometheus-telegram-bot**, **grafana-load-dashboards**, **prometheus-load-rules**)
- Используем production `ClusterIssuer` для **cert-manager** где можем

Мониторинг

- Prometheus operator
- Частично изменяем правила алертинга (например, **CPUThrottlingHigh**)
- Добавляем свои правила (**nginx-ingress**, **node autoscaling**, etc...)
- При установке добавляем dashboards для Grafana из JSON
- Настраиваем уведомления в telegram

> Многие сервисы (**prometheus**, **alertmanager**, etc...) не имеют встроенной авторизации. Поэтому закрываем их **oauth2-proxy**.

Логирование

- Набор решений вокруг **ElasticSearch**
- Небольшой объем логов
- Нет общего ElasticSearch, поэтому используем установку вида "Кластер EFK на кластер Kubernetes"
- Loki как легковесная альтернатива (предлагаем выбор исходя из потребностей)

Backlog по инфраструктуре

- Мониторинг внешних endpoint кластера (доступность, время ответа, ssl сертификаты)
- Добавление **HashiCorp Vault** как решения для хранения секретов
- Долгосрочное хранение метрик Prometheus (смотрим на remote storage, но пока нет явной необходимости)
- **Distributed Tracing**
- Назревает необходимость в **Service Mesh**
- Тестирование алертов (**promtool**)
- Генерация Dashboard для Grafana из кода (**grafonnet**)
- Отчуждение модулей Terraform из проектов
- Резервное копирование БД (в дополнение к встроенному в Cloud SQL)