Установка кластера Kubernetes

Докладчик: Илья Крылов
Дата: июнь 2020

План лекции

1. Обзор используемых инструментов
2. Критерии выбора размера кластера и его узлов
3. Схема будущего кластера
4. Создание виртуальных машин и установка кластера
5. Мониторинг
6. Управление логами
7. Backup/restore

Yandex Cloud

Официальный сайт: https://cloud.yandex.ru/

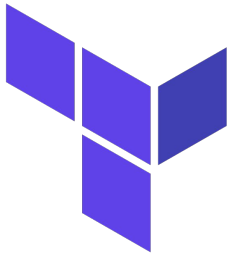
Облачная платформа от Яндекса.

Предоставляет обширный набор сервисов: виртуальные машины, различные базы данных и хранилища, Kubernetes и многое другое.

Для работы с платформой необходимо создать свое “облако” на официальном сайте. При регистрации выдается грант на несколько тысяч рублей для ознакомления с платформой.

Terraform

Официальный сайт: https://www.terraform.io/



Инструмент для декларативного управления инфраструктурой различных облачных провайдеров (Infrastructure as Code).

Позволяет автоматизировать создание, обновление и удаление необходимых ресурсов.

Для использования необходимо установить клиент: https://learn.hashicorp.com/terraform/getting-started/install

Kubespray

Официальный сайт: https://kubespray.io/

Ansible playbook для установки (и сопровождения) кластера Kubernetes (далее k8s).

Для использования необходимо установить Ansible: https://www.ansible.com/resources/get-started

Рекомендуется сделать свой fork репозитория и периодически его обновлять из основного.

https://github.com/kubernetes-sigs/kubespray

Дополнительные инструменты

kubectl – для управления k8s-кластером:
https://kubernetes.io/docs/tasks/tools/install-kubectl/

Helm – для установки Helm-пакетов:
https://helm.sh/docs/intro/install/

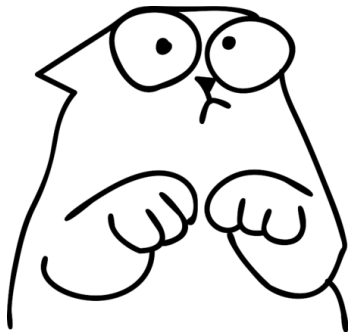
jq - консольная утилита для работы с JSON-данными:
https://stedolan.github.io/jq/

Выбор размера и количества узлов

Один большой кластер или несколько небольших кластеров?

Несколько мощных worker-узлов или много небольших?

Сколько master-узлов необходимо?

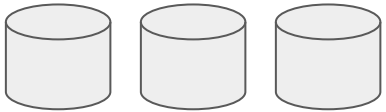


Выбор количества кластеров

	
Плюсы: ● Проще администрировать ● Дешевле при прочих равных ● Оптимальный уровень утилизации ресурсов **Минусы:** ● Единая точка отказа ● Недостаточный уровень изоляции ● Предоставление доступа к ресурсам вне namespace’ов ● Максимальный размер кластера ограничен	**Плюсы:** ● Высокий уровень изоляции ● Выше общая отказоустойчивость ● Возможность более тонкой настройки кластера ● Гибкость в предоставлении доступа **Минусы:** ● Сложнее администрировать ● Дороже при прочих равных ● Менее эффективная утилизация ресурсов

Рекомендуемый подход в рамках одной компании:
Отдельный кластер для production и отдельный кластер для остального (test, dev).

Выбор количества и размера worker-узлов

	
Плюсы: ● Проще администрировать ● Дешевле при прочих равных ● Возможность запускать требовательные нераспределенные приложения **Минусы:** ● Большое количество pod’ов на одном узле (желательно не более 110 pod’ов на одном узле) ● Меньший уровень реплицирования ● Меньше отказоустойчивость ● Большой шаг при авто-масштабировании	**Плюсы:** ● Выше отказоустойчивость ● Больший уровень реплицирования **Минусы:** ● Выше нагрузка на кластер, больше накладных расходов на системные службы и сервисы Kubernetes ● Менее эффективная утилизация ресурсов ● Меньше возможностей запускать требовательные нераспределенные приложения

Рекомендуемый подход:
Зависит от сценариев применения. Есть возможность совмещать различные типы узлов в одном кластере.

Сколько master-узлов необходимо?

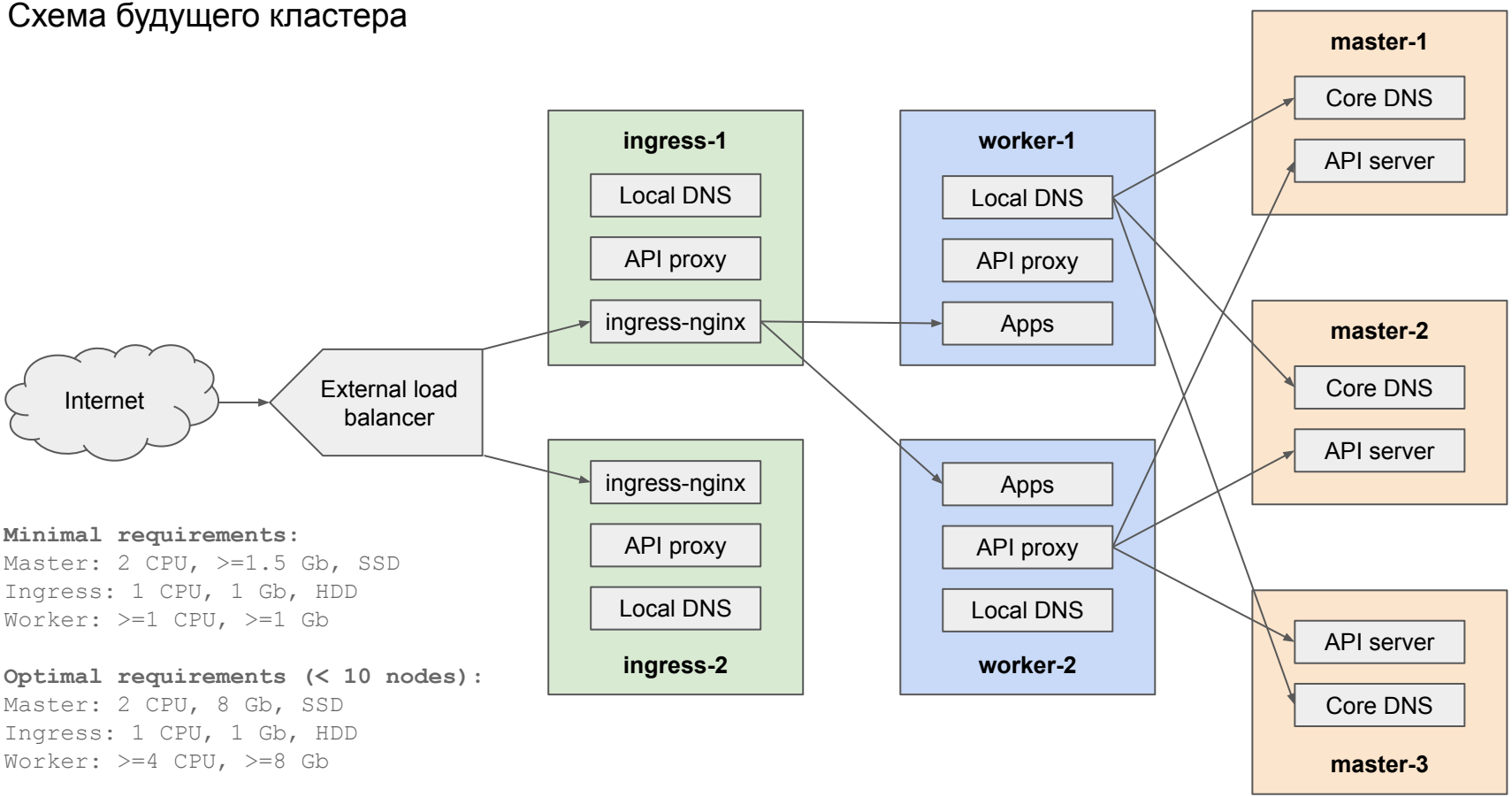
Для отказоустойчивого кластера необходимо **минимум 3 master-узла**
(для обеспечения кворума etcd).

Их мощность зависит от размера самого кластера – чем больше в кластере узлов, тем мощнее должны быть master-узлы.

Набор рекомендаций от авторов Kubernetes:
https://kubernetes.io/docs/setup/best-practices/cluster-large/#size-of-master-and-master-components

Если etcd планируется устанавливать на master-узлы, то необходимо обеспечить высокую скорость дисковой подсистемы (например, использовать SSD вместо HDD).

Схема будущего кластера



Minimal requirements:
Master: 2 CPU, >=1.5 Gb, SSD
Ingress: 1 CPU, 1 Gb, HDD
Worker: >=1 CPU, >=1 Gb

Optimal requirements (< 10 nodes):
Master: 2 CPU, 8 Gb, SSD
Ingress: 1 CPU, 1 Gb, HDD
Worker: >=4 CPU, >=8 Gb

Подготовка проекта: Kubespray

Клонируем репозиторий лекции:

```
$ git clone git@git.cloud-team.ru:lections/kubernetes_setup.git
```

Отдельно клонируем репозиторий Kubespray
(fork с небольшим фиксом для поддержки Ubuntu 20.04):

```
$ cd kubernetes_setup
$ git clone git@git.cloud-team.ru:ansible-roles/kubespray.git \
  kubespray -b cloudteam
```

Устанавливаем зависимости Kubespray:

```
$ sudo pip3 install -r kubespray/requirements.txt
```

Подготовка проекта: Terraform

Заполняем настройки для Terraform:

```
$ cp terraform/private.auto.tfvars.example terraform/private.auto.tfvars
$ vim terraform/private.auto.tfvars
```

Необходимо заполнить следующие значения:

- **`yc_token`** – OAuth-токен для доступа к API (https://cloud.yandex.ru/docs/iam/concepts/authorization/oauth-token)
- **`yc_cloud_id`** – ID облака (скопировать из консоли управления)
- **`yc_folder_id`** – ID каталога (скопировать из консоли управления)

Создание облачных ресурсов и установка кластера

```
$ bash cluster_install.sh
```

Скрипт выполняет следующие действия:

1. Запускает Terraform для создания необходимой инфраструктуры (она описана в файле “terraform/k8s-cluster.tf”).
2. Получает IP-адреса созданных виртуальных машин и генерирует файл hosts.ini для ansible-инвентаря.
3. Запускает Kubespray playbook со сгенерированным инвентарем для установки кластера на подготовленные сервера.

Состав playbook’ов в Kubespray

- **`cluster.yml`** – первоначальная установка кластера, обновление кластера, а также добавление новых master-узлов.
- **`recover-control-plane.yml`** – аварийное восстановление “control plane” (управляющего контура) кластера в случае сбоев master-узлов.
- **`remove-node.yml`** – удаление указанного узла кластера.
- **`reset.yml`** – “сброс” узла/узлов (удаление служб, контейнеров и т.д).
- **`scale.yml`** – добавление нового узла (не master-узла!).
- **`upgrade-cluster.yml`** – graceful-обновление версий в кластере (с поочередным выводом узлов из кластера и возвратом обратно).

`Важно!` Перед применением необходимо ознакомиться с документацией.

Тестирование кластера

Копируем конфигурационный файл в каталог по умолчанию:

```
$ mkdir -p ~/.kube
$ cp kubespray/inventory/mycluster/artifacts/admin.conf ~/.kube/config
```

Проверяем доступность кластера:

```
$ kubectl get nodes
$ kubectl get pods -A
```

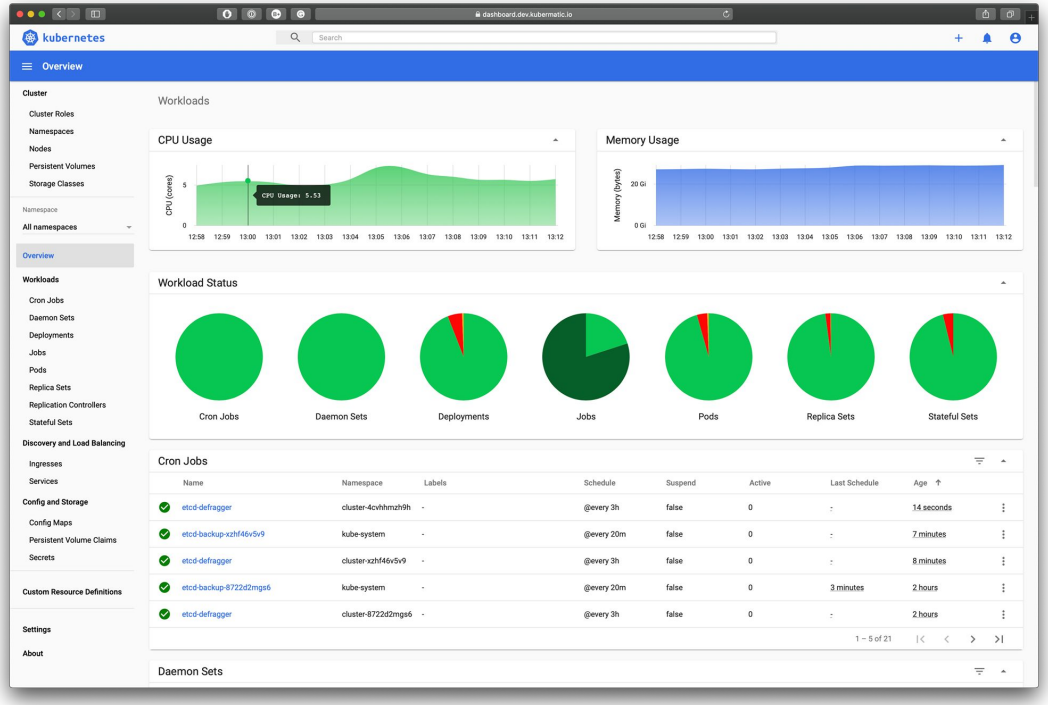
Добавляем сгенерированные хосты в наш локальный hosts-файл:

```
$ sudo sh -c "cat kubespray_inventory/etc-hosts >> /etc/hosts"
```

Запускаем тестовое приложение:

```
$ kubectl apply -f manifests/test-app.yml
$ curl hello.local
Hello from my-deployment-784598767c-7gjjs
```

Мониторинг кластера: Kubernetes Dashboard



Источник изображения: https://github.com/kubernetes/dashboard

Мониторинг кластера: Kubernetes Dashboard

Установим Kubernetes Dashboard:

```
$ helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
$ helm install --namespace monitoring --create-namespace -f manifests/dashboard-values.yml \
  kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard
$ kubectl apply -f manifests/dashboard-admin.yml
$ kubectl -n monitoring describe secret \
  $(kubectl -n monitoring get secret | grep admin-user | awk '{print $1}')
$ kubectl port-forward -n monitoring $(kubectl get pods -n monitoring \
  -l "app.kubernetes.io/name=kubernetes-dashboard" -o jsonpath="{.items[0].metadata.name}") 9090
```

В настройках Helm-чарта (файл “manifests/dashboard-values.yml”) включаем вход по http (а не по https).
В файле “manifests/dashboard-admin.yml” мы создаем service account для входа.

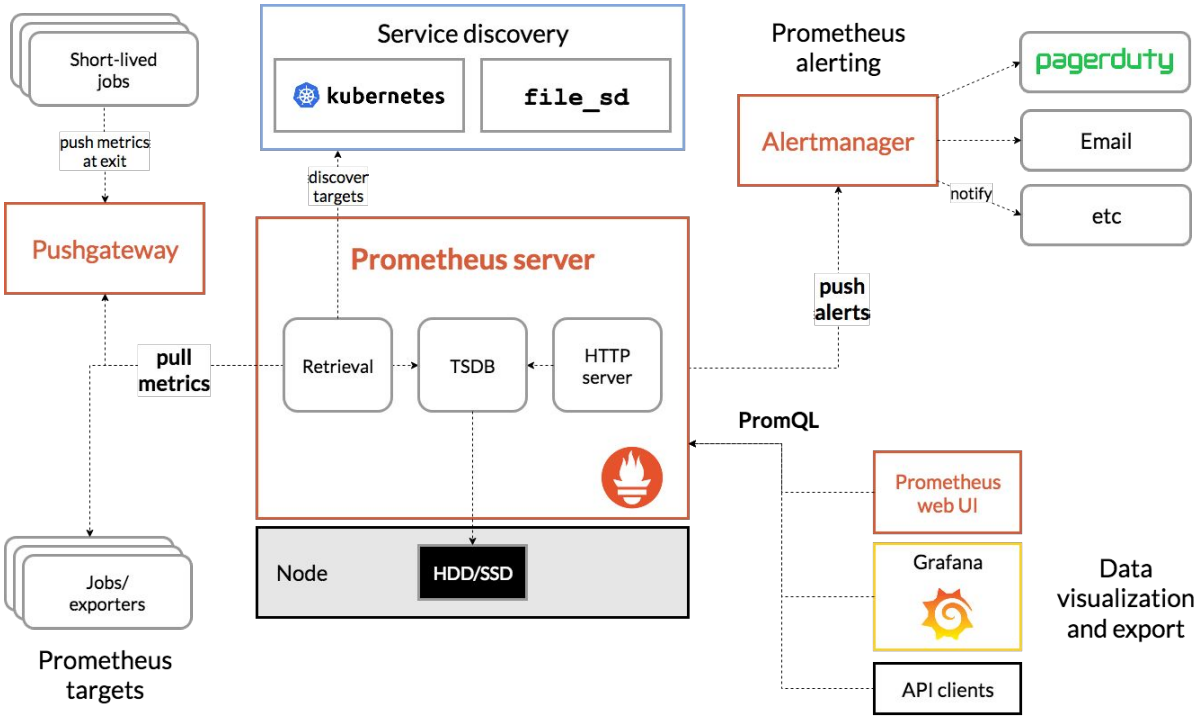
Web UI будет доступен по ссылке: http://localhost:9090.
Для аутентификации необходимо использовать token из предпоследней команды.

Мониторинг кластера: Prometheus + Grafana

Prometheus – система мониторинга, обладающая возможностями тонкой настройки метрик. Ключевые компоненты системы – это база данных временных рядов и инструменты для сбора метрик.

Grafana – инструмент с открытым исходным кодом для визуализации и анализа данных.

Мониторинг кластера: Prometheus + Grafana



Источник изображения: https://prometheus.io/docs/introduction/overview/

Мониторинг кластера: Prometheus + Grafana

Установим Prometheus с помощью Helm-чарта:

```
$ helm install --namespace monitoring --create-namespace \
  -f manifests/prometheus-values.yml prometheus stable/prometheus
```

В файле “manifests/prometheus-values.yml” указаны tolerations для pod’ов, собирающих метрики, а также указан ingress host.

Prometheus web UI будет доступен по ссылке: http://prometheus.local.

Попробуем выполнить несколько запросов с помощью PromQL.

Количество ОЗУ, используемой контейнерами Kubernetes:

```
SUM (container_memory_working_set_bytes{image!=""} / 1024 / 1024) BY (instance)
```

Использование CPU по всем узлам:

```
100 - AVG BY (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m]) * 100)
```

Мониторинг кластера: Prometheus + Grafana

Установим Grafana с помощью Helm-чарта:

```
$ helm install --namespace monitoring --create-namespace \
  -f manifests/grafana-values.yml grafana stable/grafana
$ kubectl get secret -n monitoring grafana \
  -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

В файле “manifests/grafana-values.yml” указан ingress host.

Grafana web UI будет доступен по ссылке: http://grafana.local.
(пользователь: admin, пароль: результат предпоследней команды).

Добавим новый источник данных с типом "Prometheus" (URL = http://prometheus-server).
Затем импортируем новый dashboard “https://grafana.com/dashboards/1621”.
В поле “Prometheus” нужно выбрать ранее созданный источник данных.

Управление логами

Простейший пример работы с логами:

```
$ kubectl logs my-deployment-784598767c-7gjjs
```

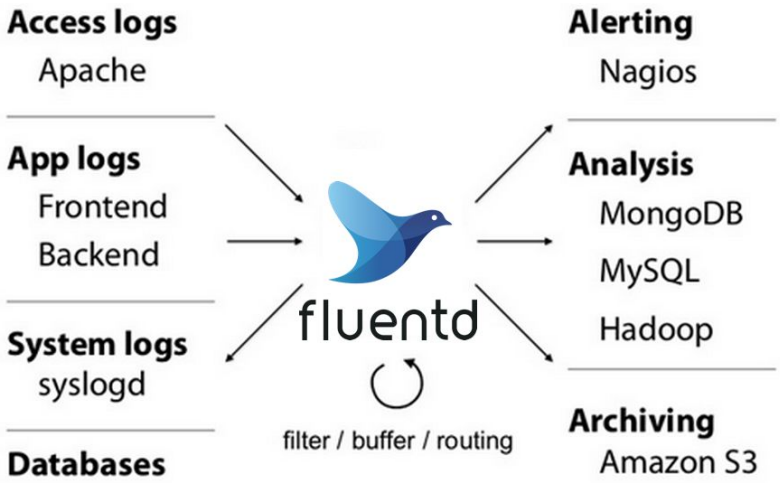
Логи контейнеров хранятся в каталогах “/var/log/containers” узлов кластера, их ротацией занимается kubelet.

Проблемы:

- Как узнать имя нужного pod’а?
- Как собрать логи сразу по группе связанных pod’ов?
- Как быть с логами самого узла (/var/log, journald, ...)?
- Как настроить пересылку логов по определенным правилам?

Управление логами: Fluentd

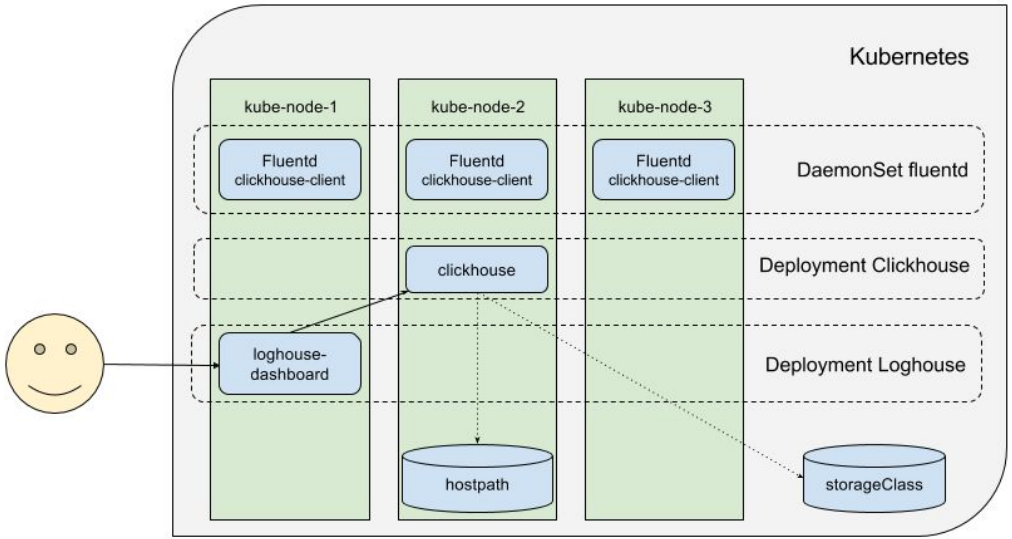
Fluentd – (https://www.fluentd.org/) инструмент для сбора и обогащения логов из различных источников, а также для маршрутизации их в различные приемники.



Источник изображения: https://docs.fluentd.org/quickstart

Управление логами: Loghouse

Loghouse - (https://github.com/flant/loghouse) инструмент просмотра логов в кластере. В качестве сборщика логов используется Fluentd, а качестве хранилища данных – Clickhouse (https://clickhouse.tech/).



Источник изображения: https://github.com/flant/loghouse

Управление логами: Loghouse

Установим Loghouse (Fluentd и Clickhouse устанавливаются вместе с ним):

```
$ helm repo add loghouse https://flant.github.io/loghouse/charts/
$ helm install --namespace loghouse --create-namespace \
  -f manifests/loghouse-values.yml loghouse loghouse/loghouse
```

В файле “manifests/loghouse-values.yml” мы задаем PVC для хранения данных, уменьшаем “container resources”, отключаем буферизацию логов, указываем ingress host, а также задаем tolerations для pod’ов с fluentd, чтобы они запускались на всех узлах кластера.

Loghouse web UI будет доступен по ссылке: http://loghouse.local (пользователь: admin, пароль: PASSWORD).

Для примера попробуем поискать логи нашего тестового приложения с помощью запроса: **~app = "my-app"**.

Backup кластера: Velero

Velero – инструмент для создания backup’ов кластера k8s.

Решает следующие задачи:

- Backup/restore приложений в кластере (как всего, так и с фильтрами).
- Мигрирование приложений в другой кластер.
- Создание копий кластера (для целей разработки или тестирования).

Состоит из серверной части, работающей внутри кластера, и консольной утилиты для управления.

Создание backup’ов может запускаться как по расписанию, так и в ручном режиме. Backup’ы могут выгружаться в различные хранилища (с помощью плагинов).

Backup кластера: Velero

Установим клиентскую утилиту Velero: https://velero.io/docs/v1.4/basic-install/
Установим серверную часть в наш кластер, указав AWS-плагин для выгрузки бекапов в S3-хранилище внутри нашего YandexCloud:

```
$ velero install \
  --provider aws \
  --plugins velero/velero-plugin-for-aws:v1.1.0 \
  --backup-location-config \
    region=ru-central1-a,s3ForcePathStyle="true",s3Url=https://storage.yandexcloud.net \
  --bucket backup-backet \
  --snapshot-location-config region=ru-central1-a \
  --secret-file kubespray_inventory/credentials-velero
```

Файл “credentials-velero” был автоматически сгенерирован ранее скриптом “generate_credentials_velero.sh”.

Backup кластера: Velero

Создадим наш первый backup:

```
$ velero backup create --selector app=my-app my-first-backup
```

Проверим статус backup'а:

```
$ velero backup describe my-first-backup
$ velero backup get
```

Удалим ранее запущенное тестовое приложение:

```
$ kubectl delete -f manifests/test-app.yml
$ kubectl get pods | grep my-deployment
```

Восстановим только что созданный backup:

```
$ velero restore create --from-backup my-first-backup
```

Проверим статус восстановления:

```
$ velero restore get
```

Удаление кластера и облачных ресурсов

```
$ bash cluster_destroy.sh
```

Скрипт удаляет все созданные облачные ресурсы
(а соответственно и сам кластер).

Дополнительная информация и источники

- **Репозиторий с примерами из лекции:** https://git.cloud-team.ru/lections/kubernetes_setup
- Репозиторий с fork'ом Kubespray: https://git.cloud-team.ru/ansible-roles/kubespray
- Документация Яндекс.Облако: https://cloud.yandex.ru/docs
- Документация Terraform: https://www.terraform.io/docs/index.html
- Документация Kubespray: https://kubespray.io/
- Документация Kubernetes Dashboard: https://github.com/kubernetes/dashboard
- Документация Prometheus: https://prometheus.io/docs/introduction/overview/
- Статья "Install Prometheus and Grafana by Helm": https://medium.com/@at_ishikawa/install-prometheus-and-grafana-by-helm-9784c73a3e97
- Документация о возможностях PromQL: https://prometheus.io/docs/prometheus/latest/querying/basics/
- Документация Fluentd: https://docs.fluentd.org/
- Документация Loghouse: https://github.com/flant/loghouse
- Документация Velero: https://velero.io/
- Статья "Architecting Kubernetes clusters — choosing a cluster size": https://itnext.io/architecting-kubernetes-clusters-choosing-a-cluster-size-92f6feaa2908
- Статья "Architecting Kubernetes clusters — choosing a worker node size": https://learnk8s.io/kubernetes-node-size

Спасибо за внимание!