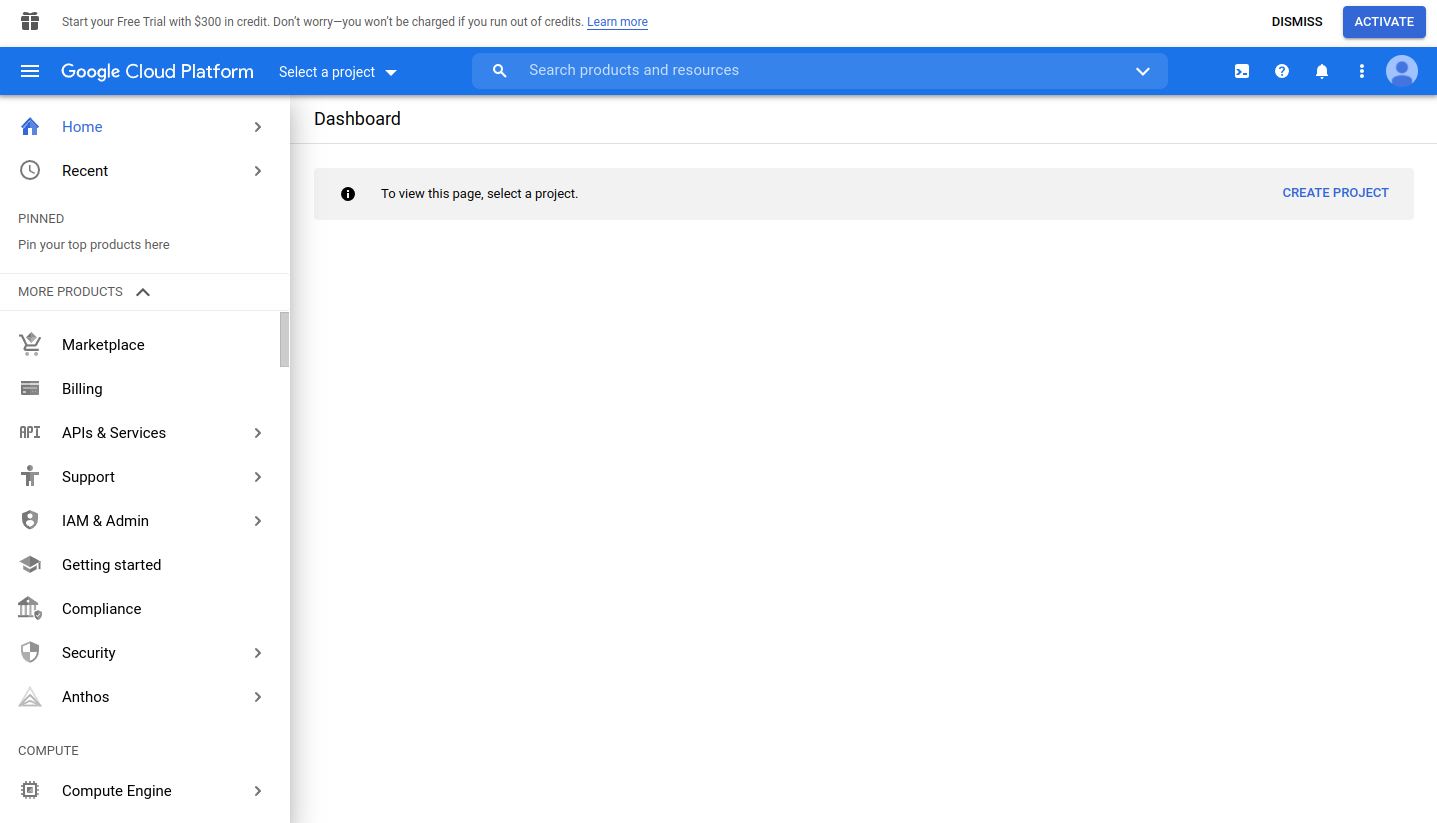
# Создание кластера Kubernetes

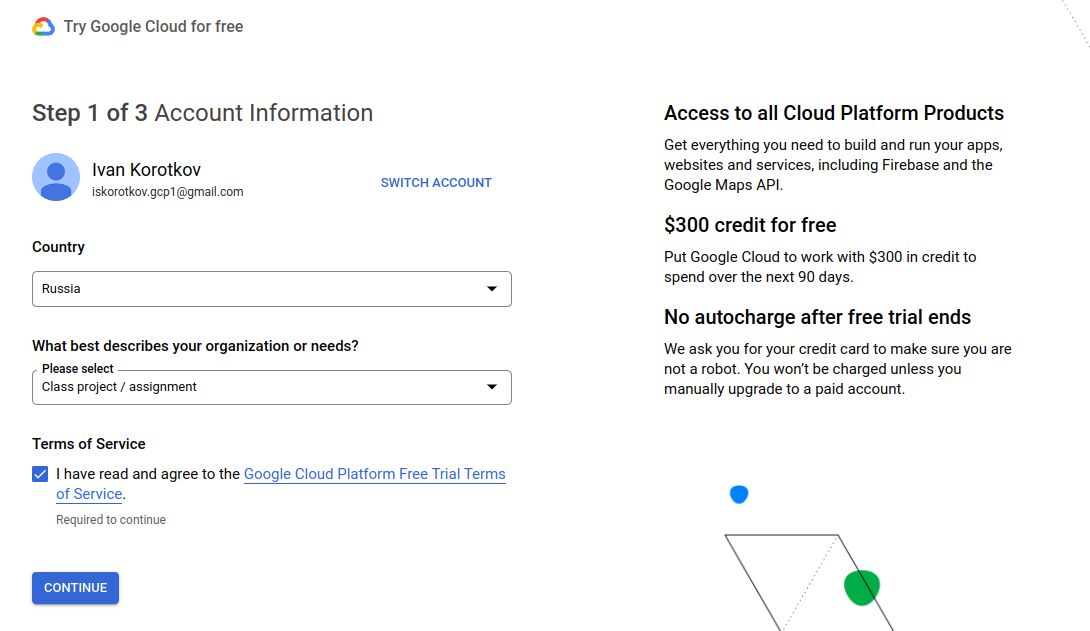
## Создание проекта в Google Cloud Platform

Создать аккаунт Google

Войти со своим аккаунтом в Google Cloud Platform (<https://console.cloud.google.com/>)

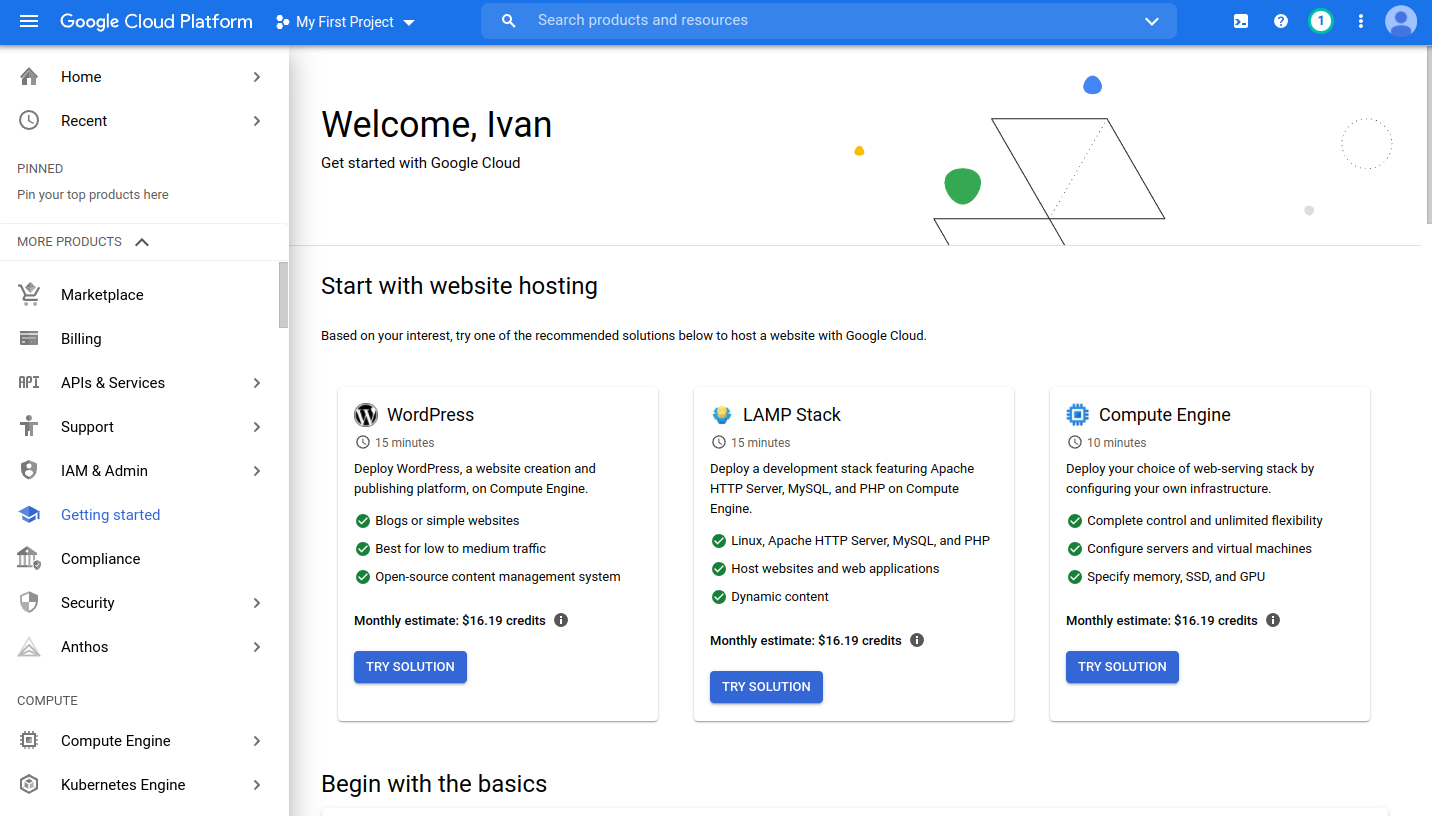


Активировать бесплатные $300 (кнопка вверху)

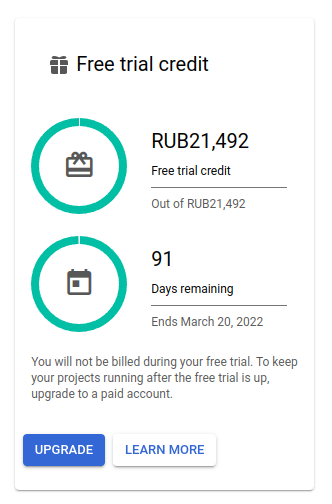


Ввести данные (в т. ч. данные карты - что поделать)

Ваш первый проект будет создан автоматически (My First Project)

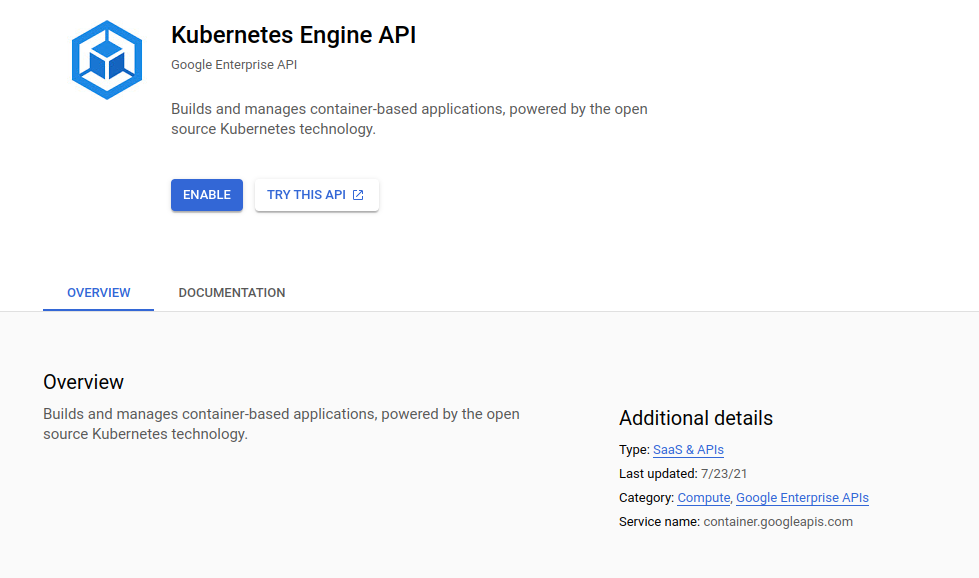


Посмотреть информацию об оставшихся и потраченных рублях вы можете на вкладке Billing (слева)

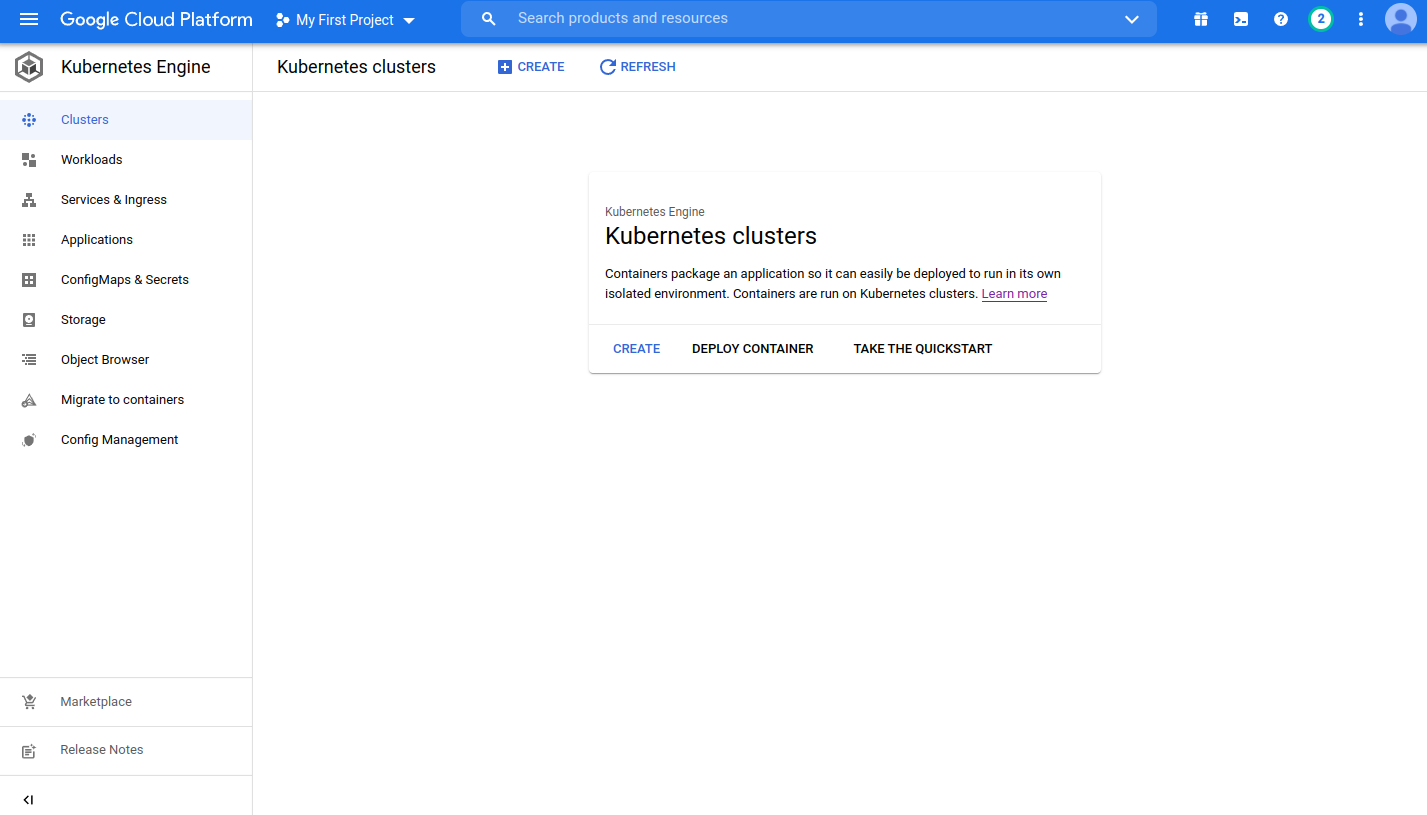


## Создание кластера в Google Kubernetes Engine (GKE)

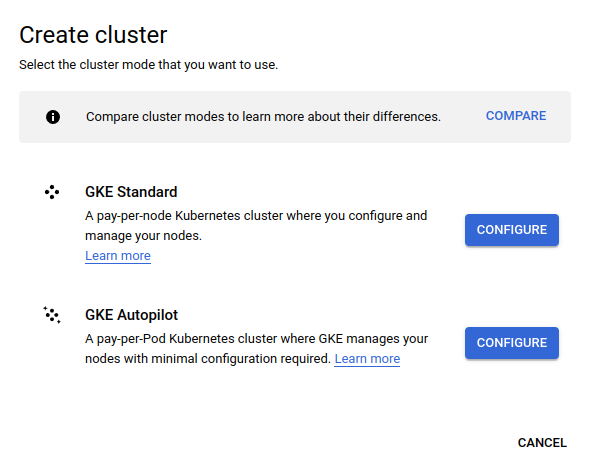
Перейдите во вкладку Kubernetes Engine (слева) и включите Kubernetes Engine API

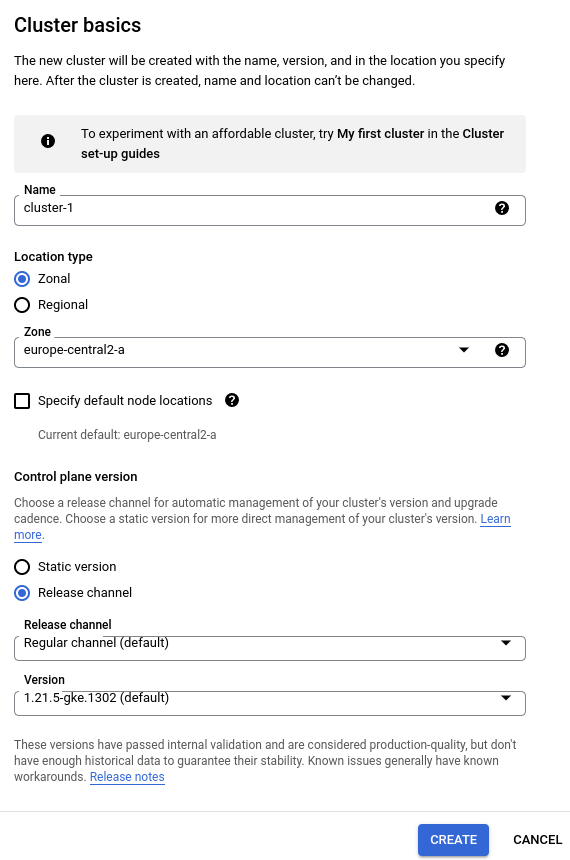


Создайте новый кластер



Выберите GKE Standart. Поменяйте регион на europe-\*. Все остальные значения можно оставить по умолчанию. При версии Kubernetes 1.22 и больше при установке Argo Workflows могут возникнуть ошибки. Если очень требуется использовать именно версию 1.22+ и ошибка возникает, сообщите мне





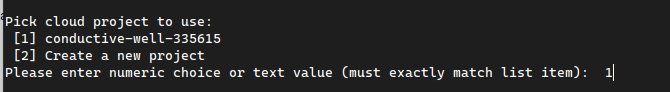
# Установка инструментов

## gcloud

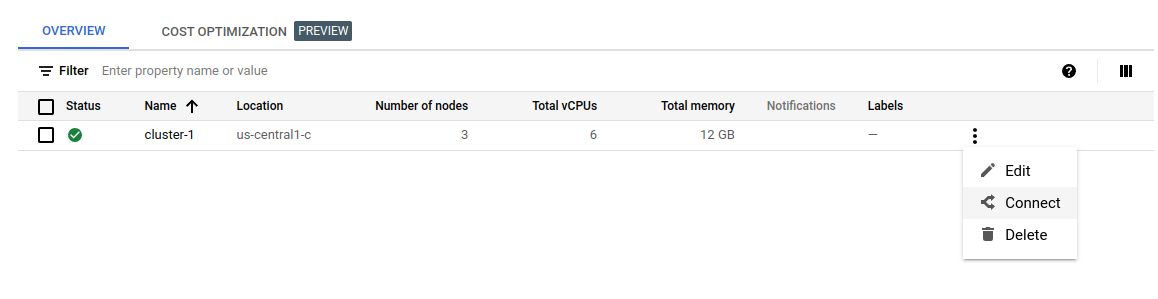
Установите gcloud (<https://cloud.google.com/sdk/docs/install>). Инструкции по установке различаются для различных платформ. Необходимо скачать, распаковать и установить gcloud в соответствии с инструкциями для вашей платформы

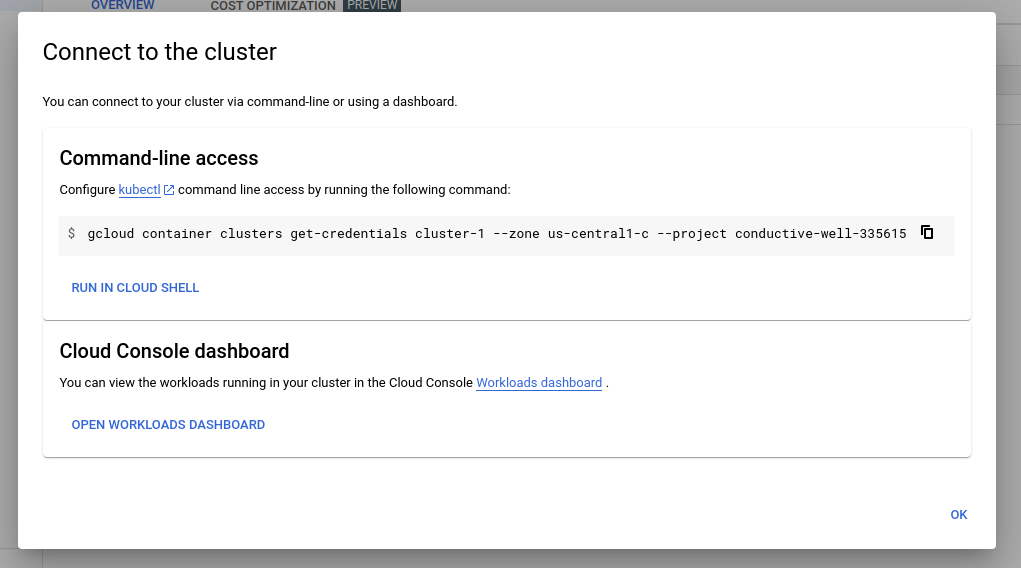
-

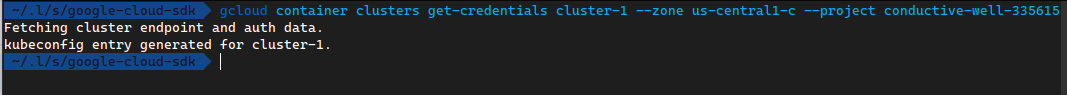
Выберите проект в gcloud



Подключитесь к своему кластеру с помощью gcloud, нажав на connect

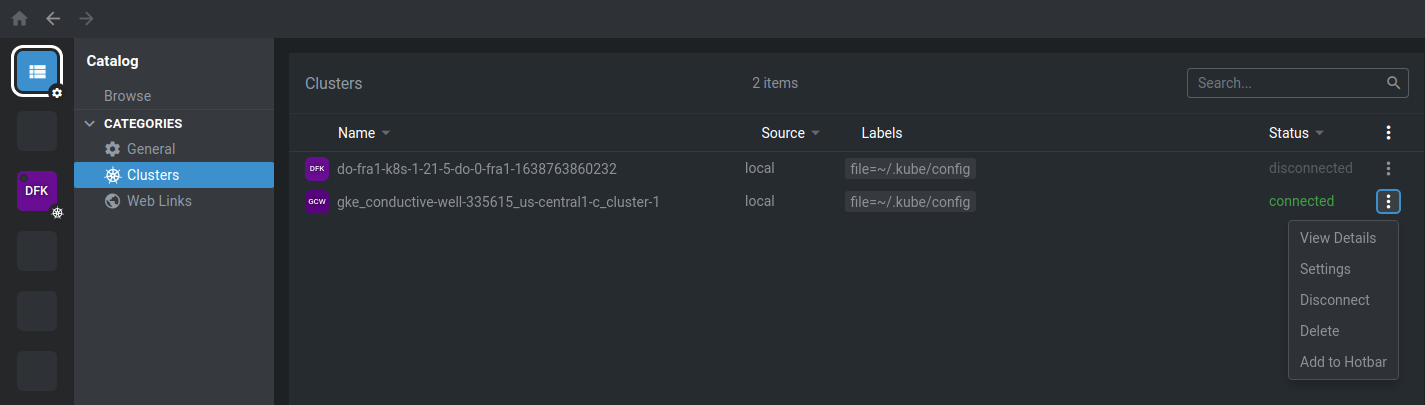


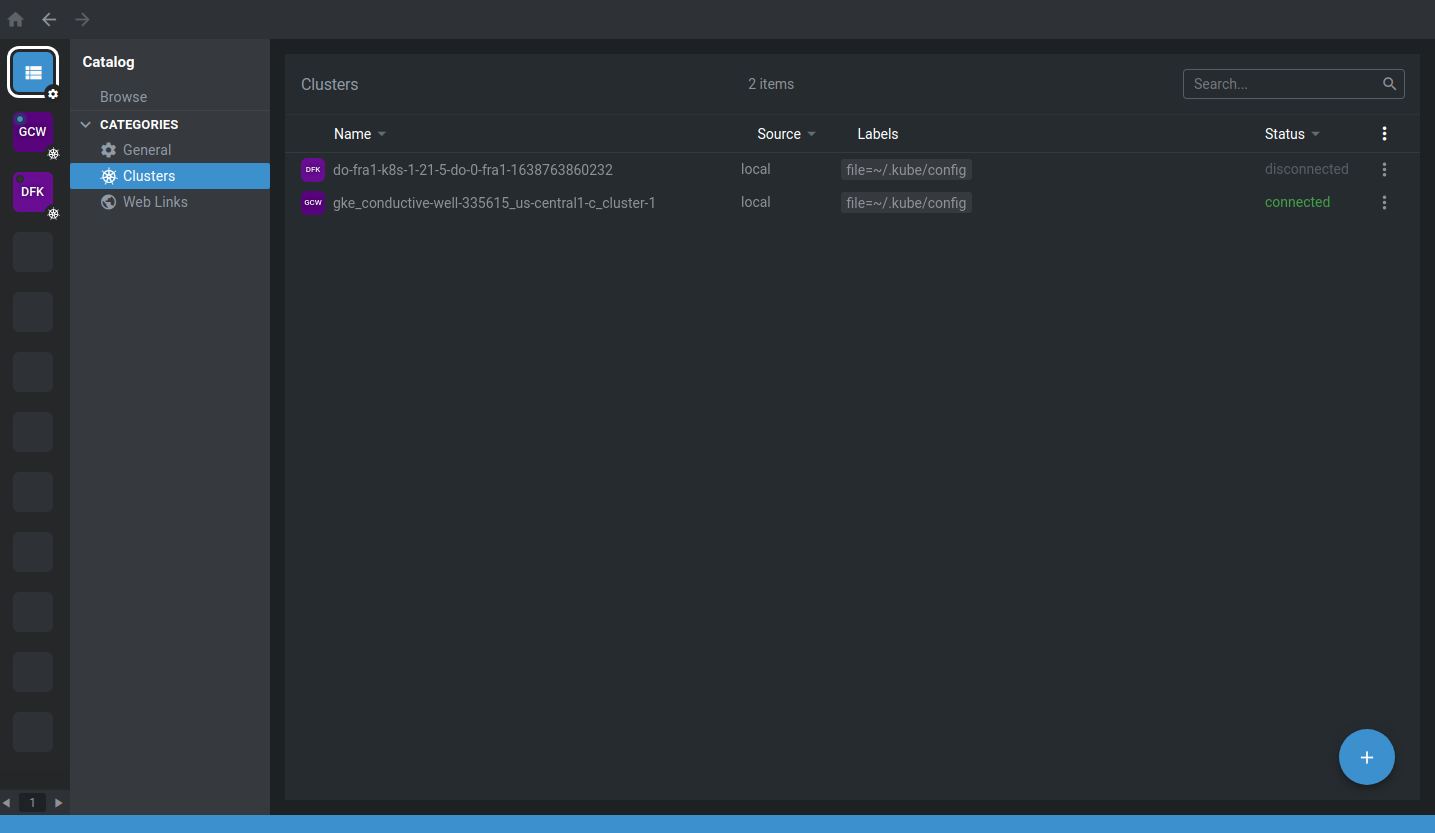




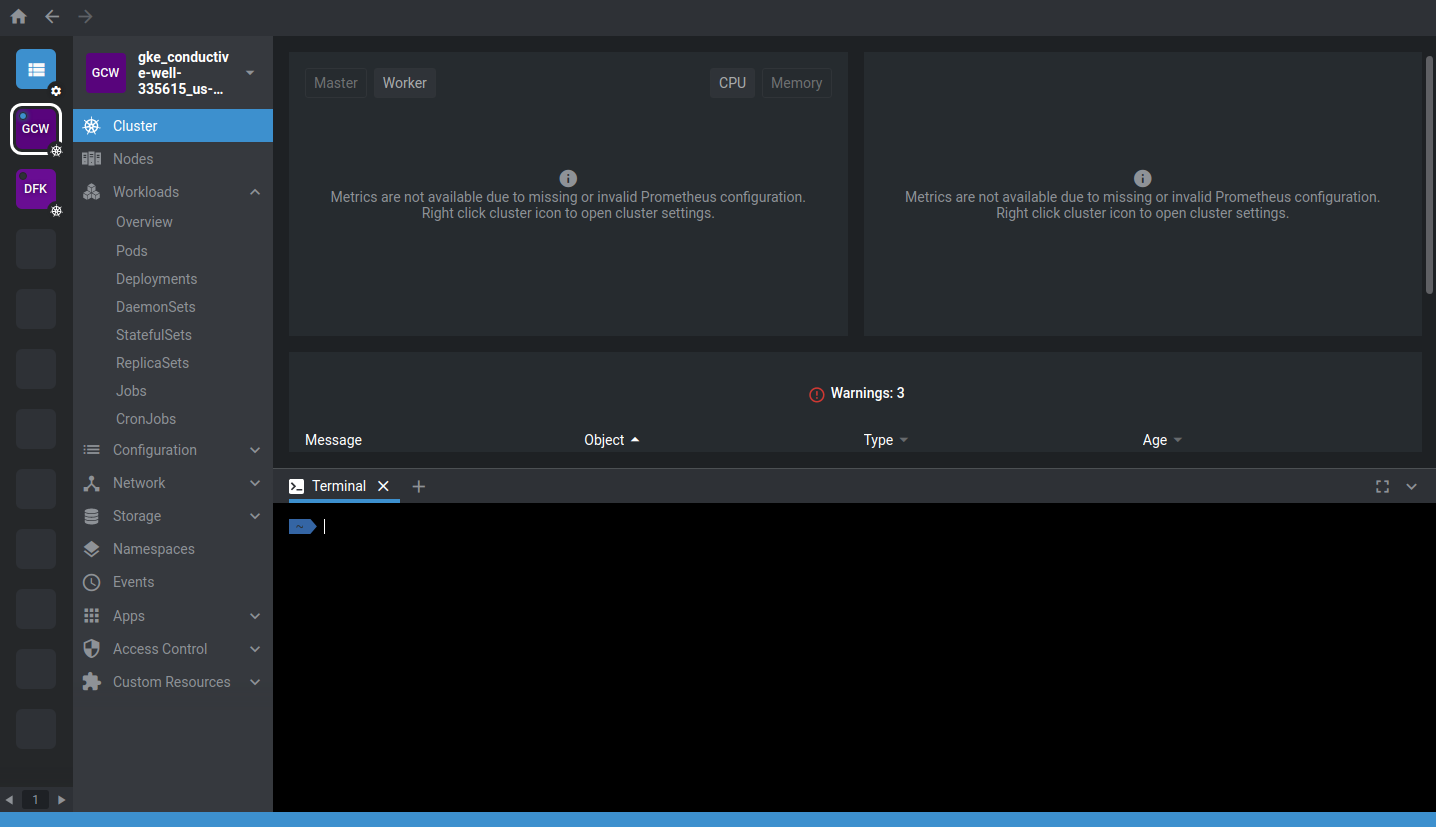
## Lens

Установите Lens - IDE for Kubernetes (<https://k8slens.dev/>). Затем в Catalog -> Clusters вы можете выбрать добавленный вами кластер. Также вы можете добавить кластер в Hotbar (узкую панель слева)





Вы можете переключаться между различными вкладками. Также вы можете открыть терминал. Утилита kubectl должна быть автоматически установлена Lens и доступна в терминале



# Установка системы тестирования

## Lutris Chaos

Установите Lutris Chaos с помощью терминала

# Install Litmus operator.

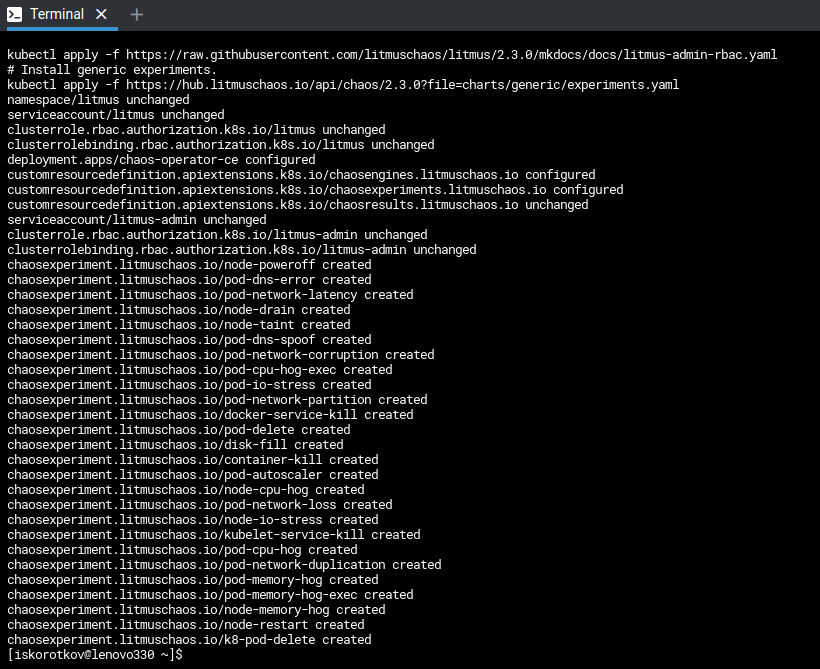
kubectl apply -f <https://raw.githubusercontent.com/litmuschaos/litmus/2.3.0/mkdocs/docs/litmus-operator-v2.3.0.yaml>

# Install service account for Litmus.

kubectl apply -f <https://raw.githubusercontent.com/litmuschaos/litmus/2.3.0/mkdocs/docs/litmus-admin-rbac.yaml>

# Install generic experiments.

kubectl apply -f <https://hub.litmuschaos.io/api/chaos/2.3.0?file=charts/generic/experiments.yaml> -n litmus



## Argo Workflows

Установите Argo Workflows с помощью терминала

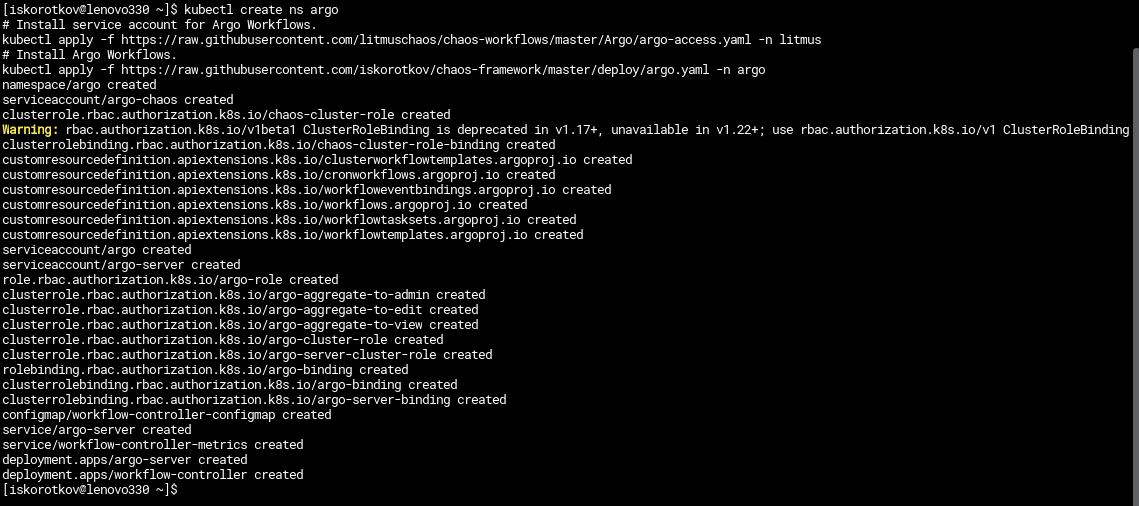
kubectl create ns argo

# Install service account for Argo Workflows.

kubectl apply -f https://raw.githubusercontent.com/litmuschaos/chaos-workflows/master/Argo/argo-access.yaml -n litmus

# Install Argo Workflows.

kubectl apply -f https://raw.githubusercontent.com/iskorotkov/chaos-framework/master/deploy/argo.yaml -n argo



## Chaos Framework

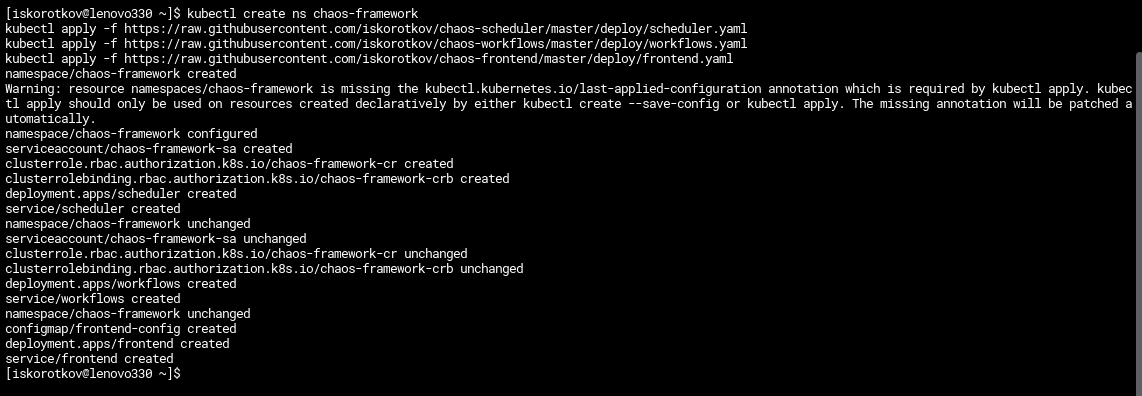
Установите Chaos Framework (мою курсовую, <https://github.com/iskorotkov/chaos-framework>) с помощью терминала

kubectl create ns chaos-framework

kubectl apply -f https://raw.githubusercontent.com/iskorotkov/chaos-scheduler/master/deploy/scheduler.yaml

kubectl apply -f https://raw.githubusercontent.com/iskorotkov/chaos-workflows/master/deploy/workflows.yaml

kubectl apply -f https://raw.githubusercontent.com/iskorotkov/chaos-frontend/master/deploy/frontend.yaml



# Устройство и принципы работы Kubernetes

## Container

В общем случае, если вам нужно запустить какое-то приложение локально, то вы можете:

1. Использовать утилиту командной строки (dotnet run)
2. Использовать кнопку или горячие клавиши в любимой IDE

Однако такое работает только в том случае, если у вас локально установлен компилятор/интерпретатор и все необходимые для работы библиотеки. Если вы попросите своего одногруппника или бабушку запустить это приложение у себя, они, вероятно, столкнутся с трудностями и не смогут запустить его просто так.

В некоторых случаях можно собрать бинарник, включающий все необходимые библиотеки, но это возможно не всегда.

Для решения этой проблемы можно использовать образы и контейнеры. Образ показывает, какой другой образ взять за основу, какие библиотеки и зависимости установить, как собрать проект и что делать при запуске. Контейнер - конкретный экземпляр образа, который непосредственно запускается.

Таким образом:

1. Сборка приложения описывается и легко воспроизводится
2. Контейнеры по собранному образу могут быть запущены где угодно, где можно запускать контейнеры

Для работы с образами и контейнерами используются инструменты Docker (<https://www.docker.com/get-started>), Podman (<https://podman.io/getting-started/>) и др.

## Pod

Документация: <https://kubernetes.io/docs/concepts/workloads/pods/>

Под - минимальная единица развертывания Kubernetes (т. е. Вы не можете запустить что-то меньше пода, например, один контейнер). Под - это “корзина” из одного или нескольких контейнеров, запускаемых всегда вместе. В простых случаях (в большинстве случаев) под состоит из единственного контейнера.

## Deployment

Документация: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

Управлять отдельными подами сложно - представьте себе запуск или обновление 50 абсолютно одинаковых подов. Для управления группой однотипных подов используются развертывания.

Развертывание - это “менеджер” подов, который стремится свести число подов к нужному числу (создавая или удаляя их). У развертывания есть шаблон, по которому он создает поды, и селектор, по которому он определяет, какие поды должны учитываться при их подсчете.

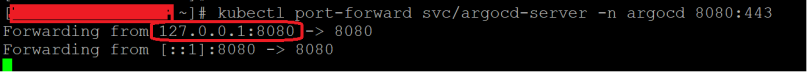
## Service

Для того, чтобы распределять трафик по нескольким подам (и не только для этого), используются сервисы. Сервис подобно развертыванию имеет селектор, показывающий, какие поды будут участвовать в балансировке нагрузки (т. е. входящих запросов). Как правило, любое взаимодействие между частями приложения проходят через сервисы.

## Port-forward

Документация: <https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster/>

По умолчанию, все сервисы внутренние, т. е. доступ к ним невозможен извне кластера (например, с вашего компьютера). Для временного доступа к конкретному сервису можно использовать port-forward - это позволит вам перенаправлять локальные порты на порты сервисов в кластере (что-то вроде туннеля от порта на вашем компьютере к порту сервиса). Например, вы можете сделать port-forward 8888:9999 для сервиса my-nginx-site, предоставляющего интерфейс через порт 9999, и тогда вы сможете взаимодействовать с ним через локальный порт 8888.



## Namespace

Документация: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>

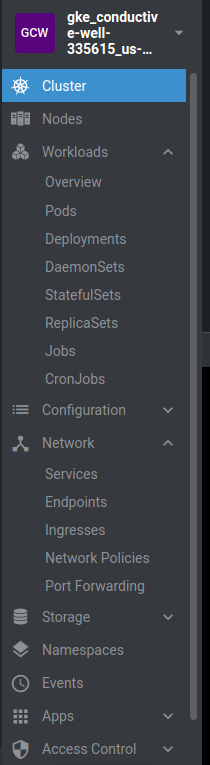
Пространства имен группируют ресурсы. В рамках одного пространства имен никакие два пода, развертывания или сервиса не могут иметь одинаковое имя (но развертывание и сервис могут иметь одинаковые имена).

# Обзор инструментов

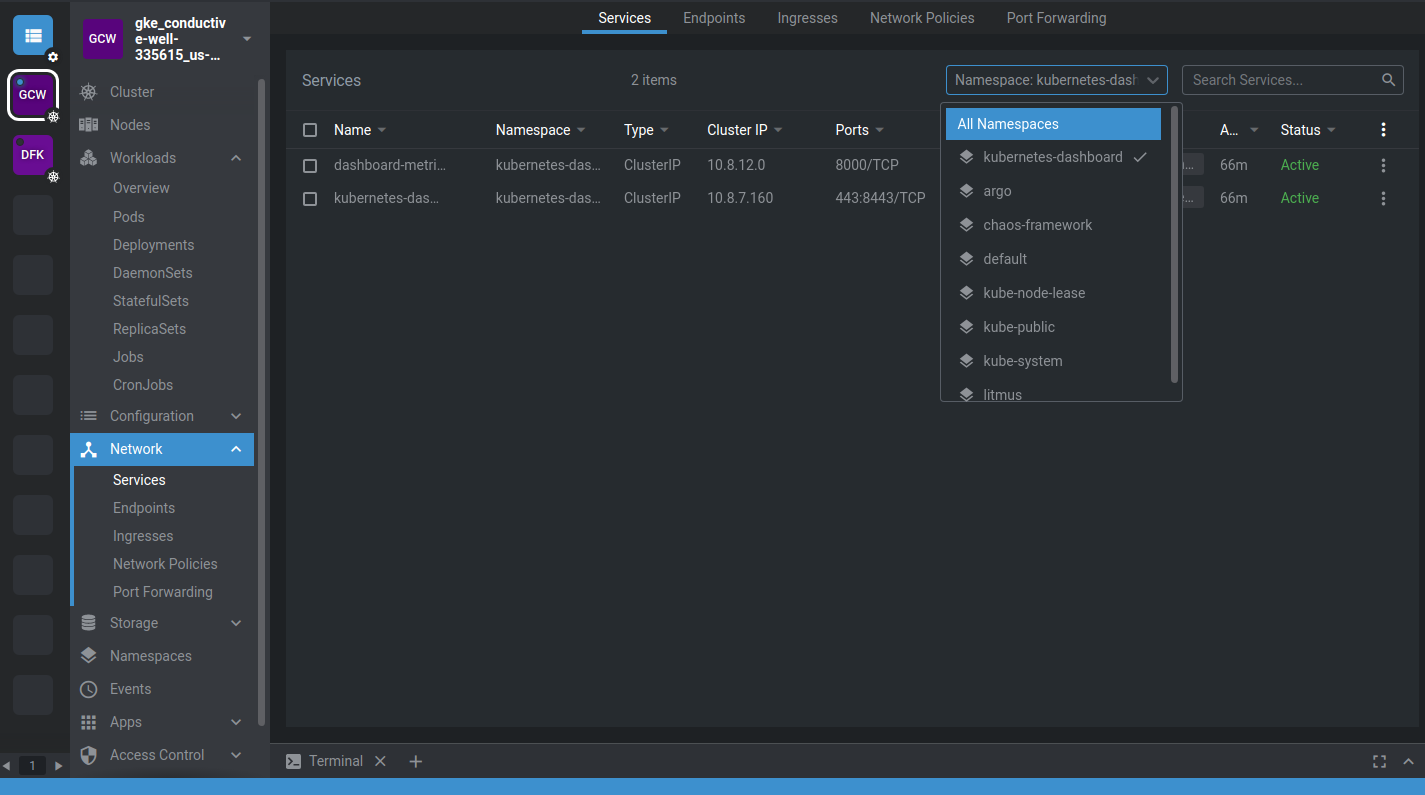
## Lens

### Обзор

Lens позволяет работать со всеми описанными видами ресурсов.

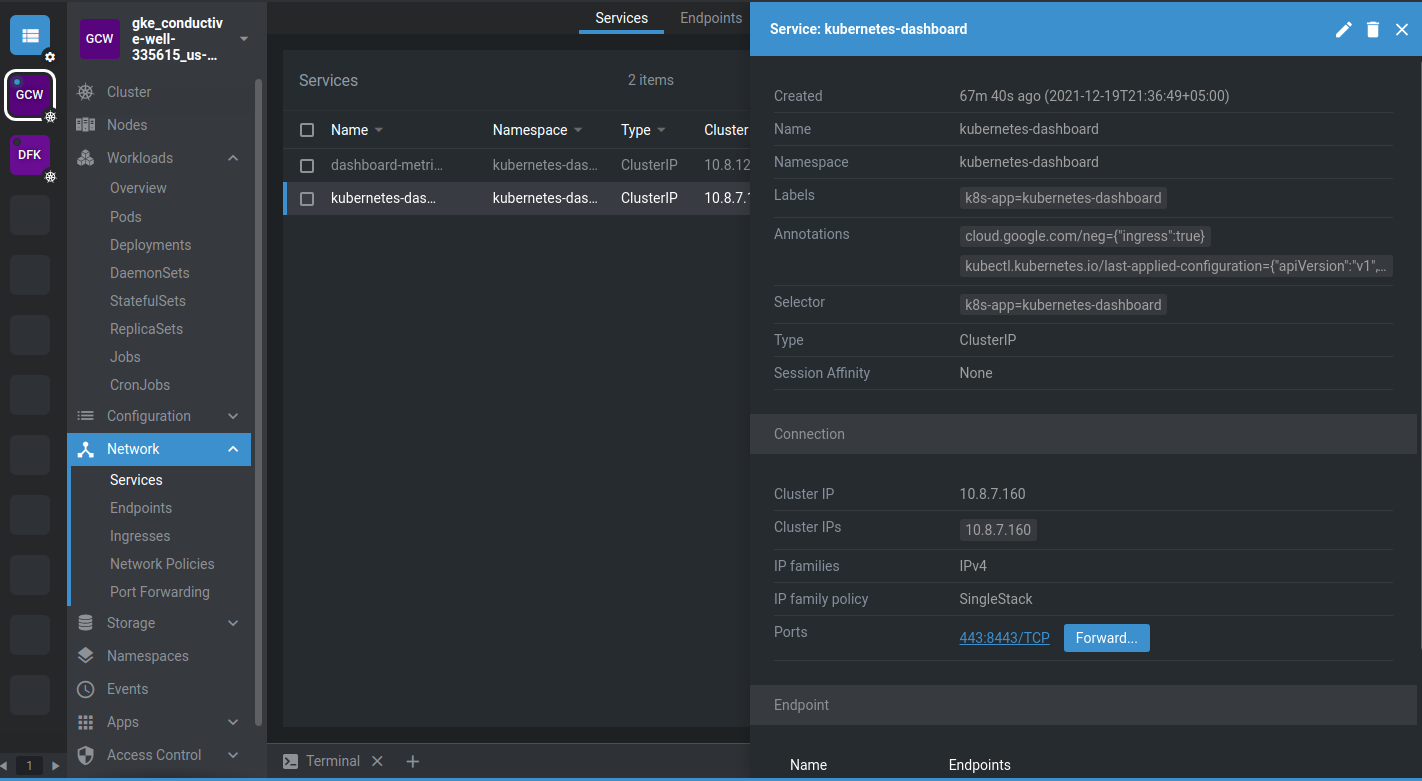


Вы можете выбрать, например, вкладку Сервисы. Далее можете выбрать пространство имен

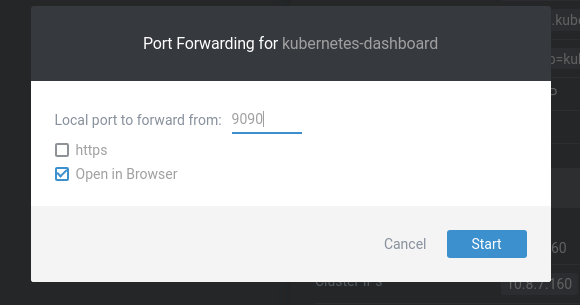


### Port forwarding

Для использования port-forward выберите сервис (на самом деле можно выбрать даже под или развертывание) и нажмите на Forward…



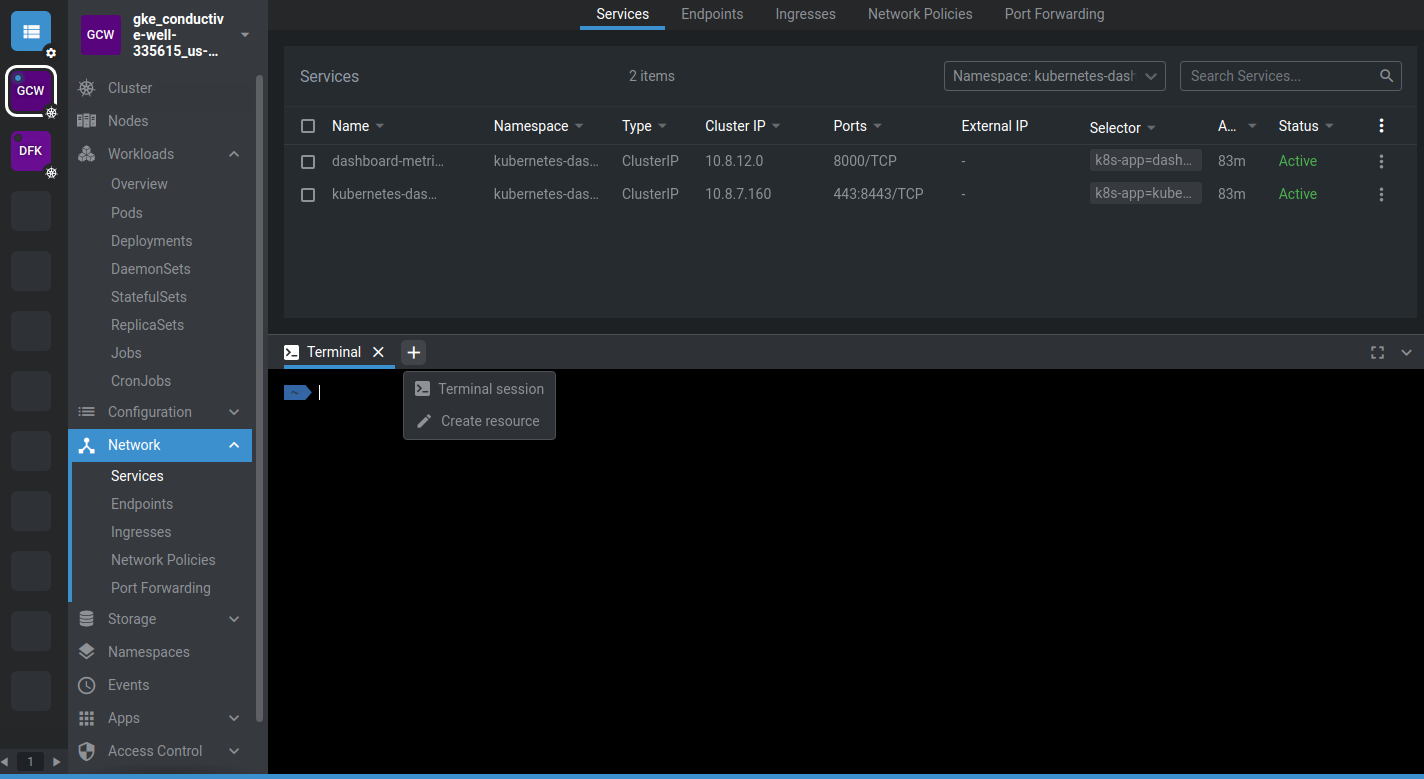
Укажите порт (если хотите конкретный порт) или оставьте пустое поле для использования случайного незанятого порта

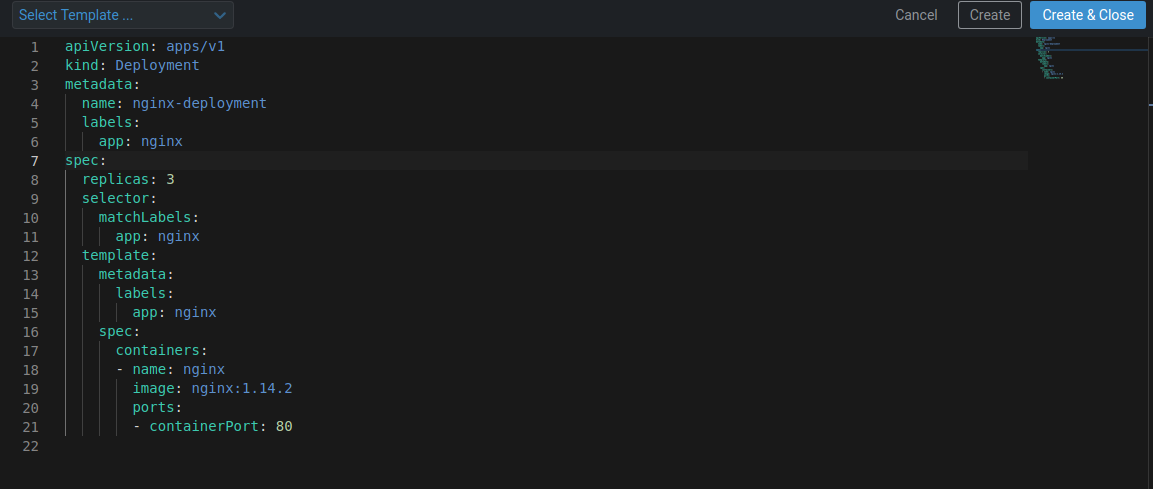


Для просмотра активных port-forward используйте вкладку Port-Forwarding.

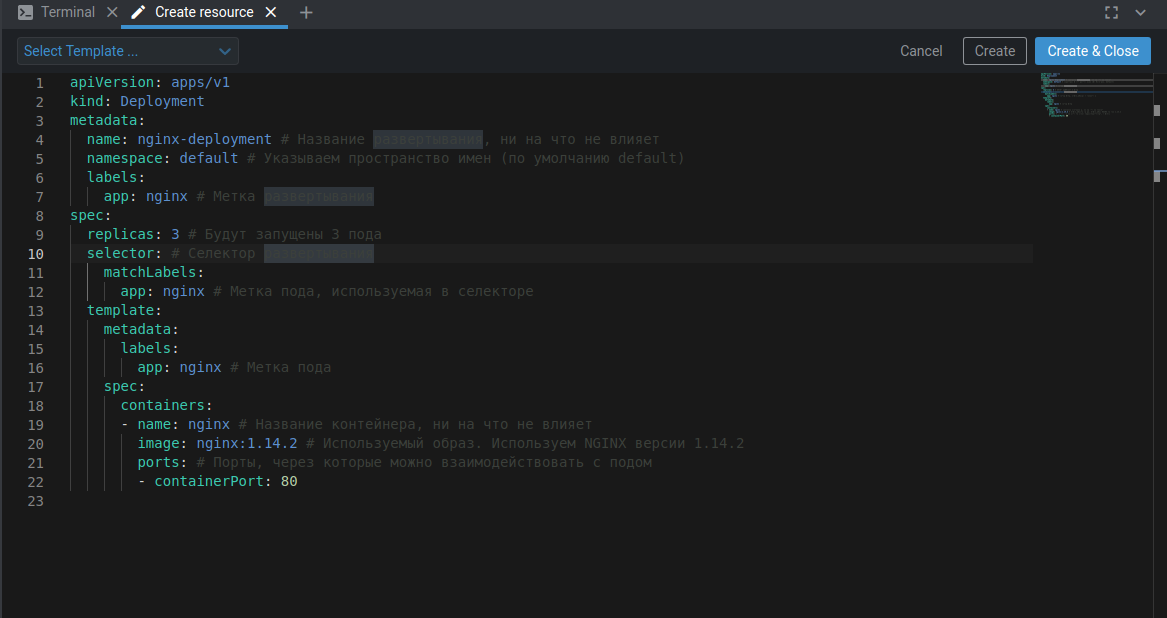
### Создание новых ресурсов

Нажмите на + -> Create resource. Выберите в качестве шаблона Deployment

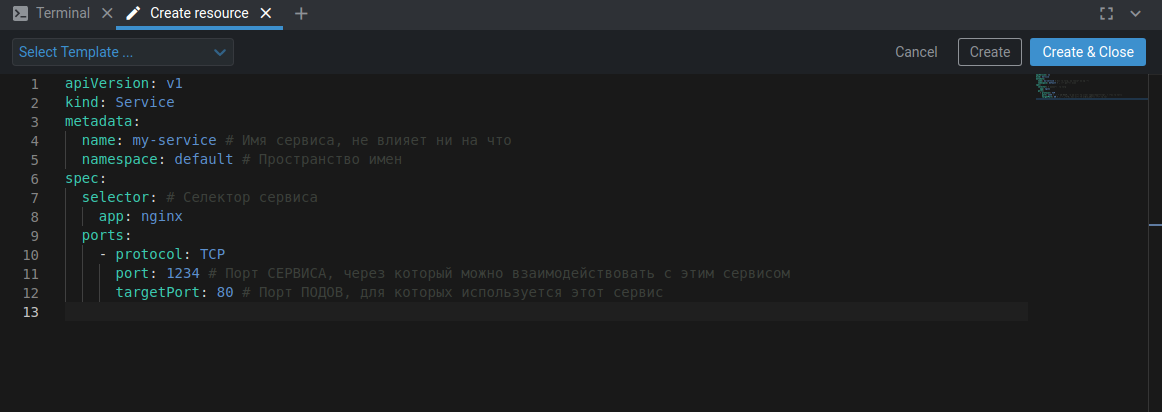




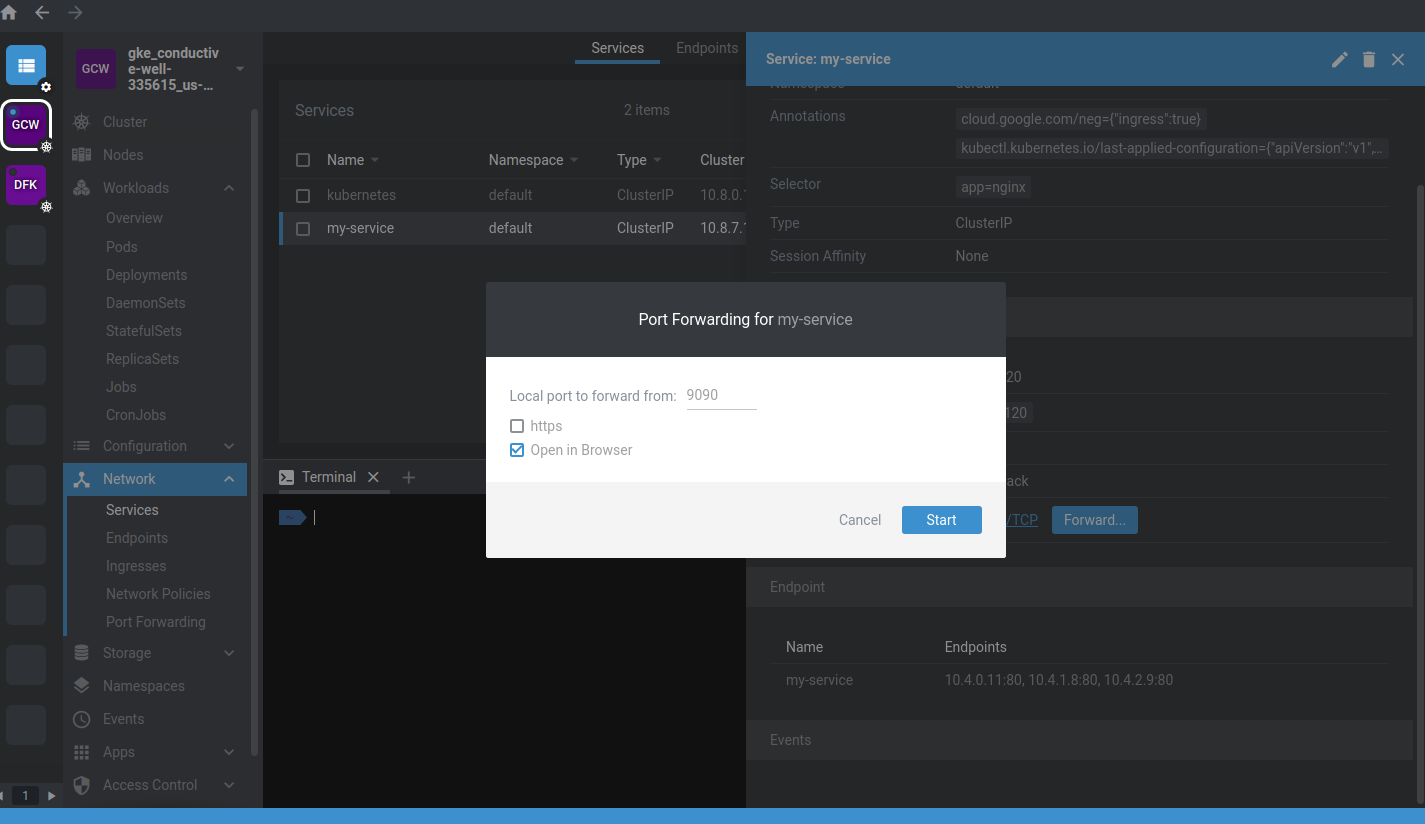
Укажите пространство имен (namespace: default) и создайте развертывание



Далее создайте сервис для данного развертывания. Укажите пространство имен, измените селектор и порты



Создайте port-forward для созданного сервиса и насладитесь минимализмом страницы с приветствием NGINX (ответ приходит от одного из трех подов)





# Запуск приложения-примера

## Описание

В рамках своей курсовой я написал реализацию алгоритма забияки ([Алгоритм забияки — Википедия](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B7%D0%B0%D0%B1%D0%B8%D1%8F%D0%BA%D0%B8)) - механизм выбора лидера в распределенной системе, в которой лидером становится процесс с наибольшим ID.

Моя курсовая работа (вы можете найти в ней главы 3.6 и 3.7 с описанием данной системы и ее тестирования): <https://1drv.ms/b/s!AvwLCFubzVg2kPw5xbmFMhsunmcmjw?e=GNHcTA>

## Запуск

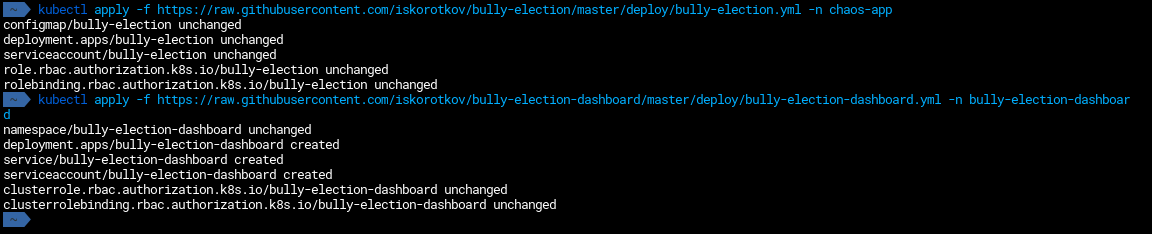
Для тестирования систему на данный момент требуется запускать в пространстве имен chaos-app. Создадим это пространство имен

kubectl create ns chaos-app

Для запуска системы требуется выполнить (-n <name> создает ресурсы из файла в указанном пространстве имен)

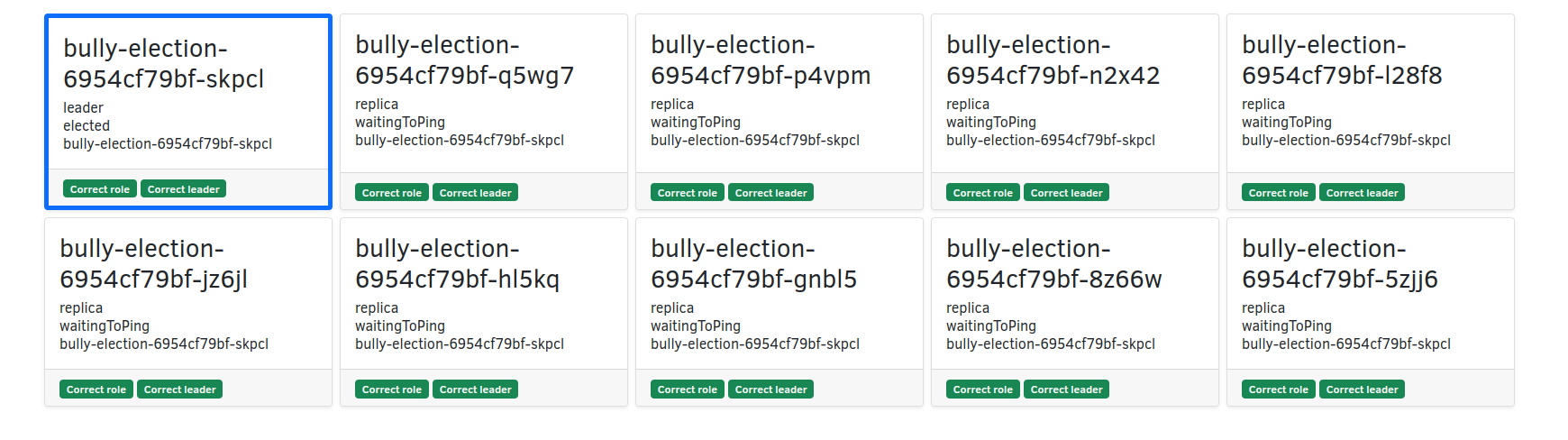
kubectl apply -f <https://raw.githubusercontent.com/iskorotkov/bully-election/master/deploy/bully-election.yml> -n chaos-app

kubectl apply -f <https://raw.githubusercontent.com/iskorotkov/bully-election-dashboard/master/deploy/bully-election-dashboard.yml> -n bully-election-dashboard



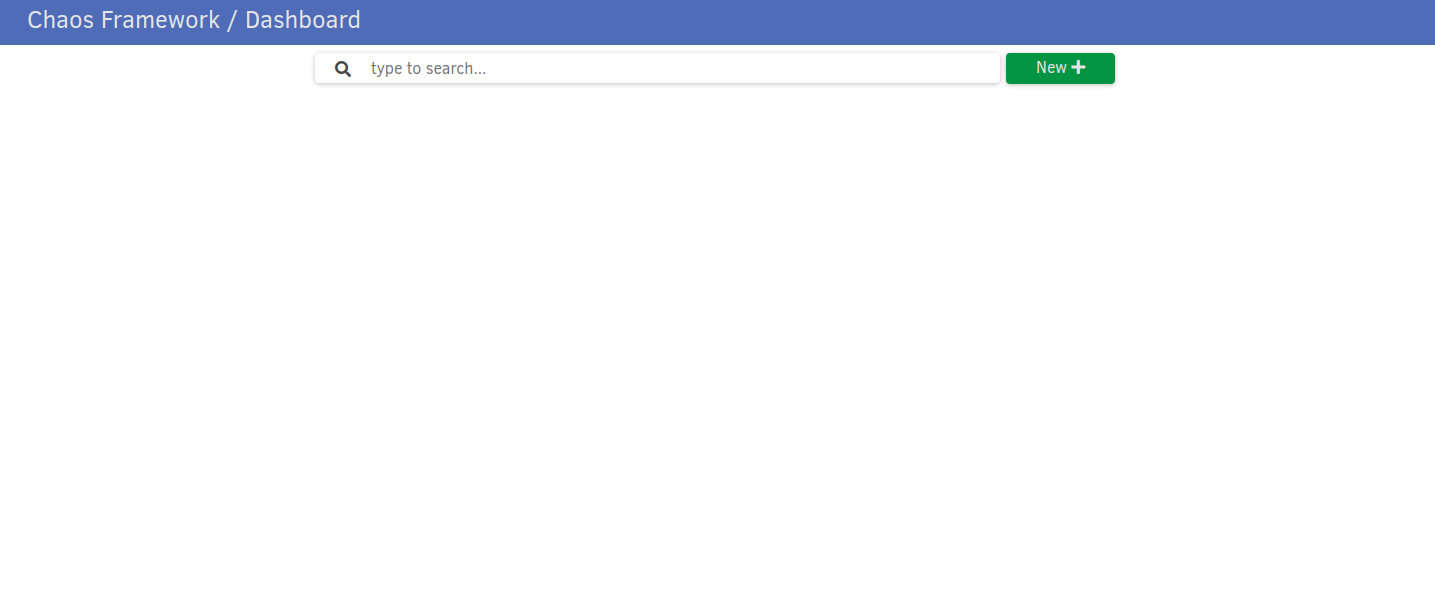
## Мониторинг

Создайте port-forward для сервиса bully-election-dashboard из пространства имен bully-election-dashboard. В браузере вы увидите дашборд с состоянием 10 реплик (по умолчанию). Первая реплика, выделенная синим контуром - текущий лидер (по крайней мере, если всё работает корректно, именно эта реплика должна быть лидером)

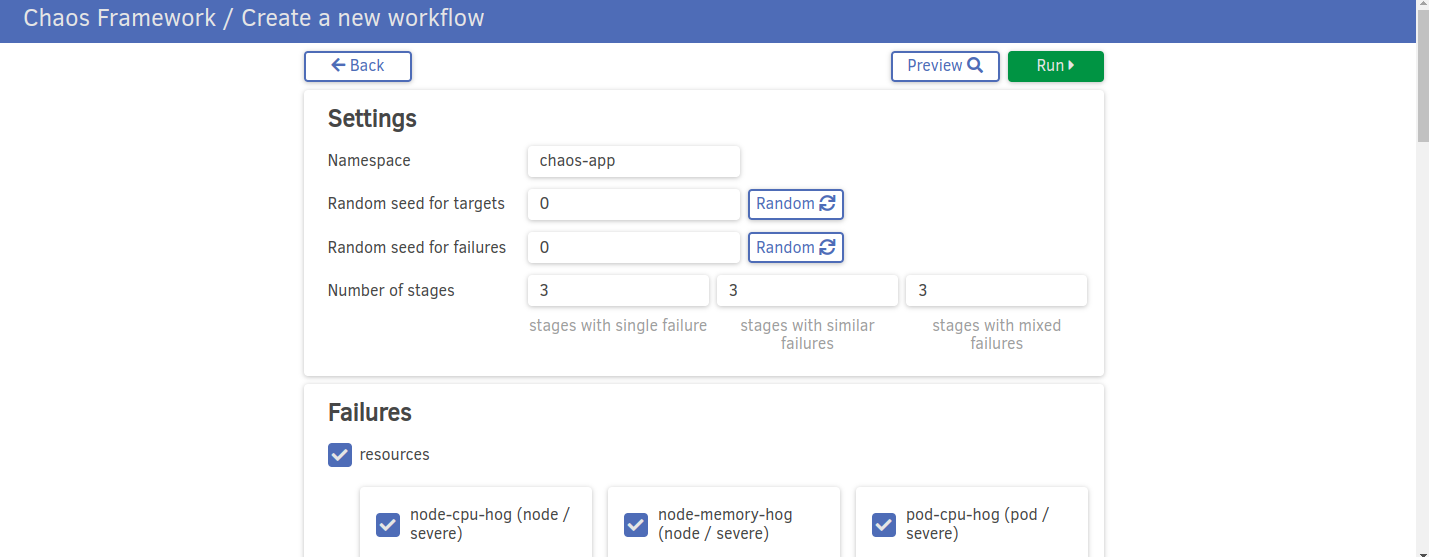


## Тест

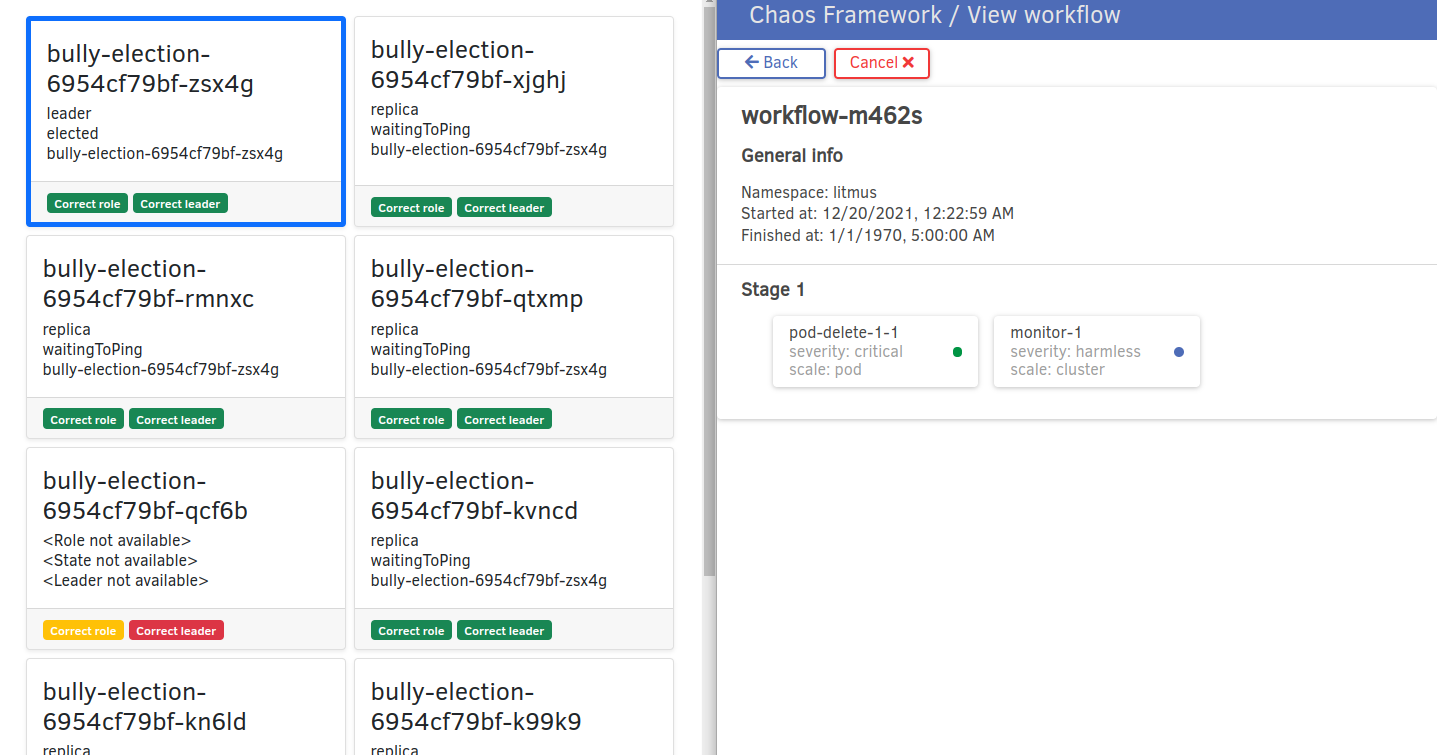
Создайте port-forward для сервиса frontend в пространстве имен chaos-framework

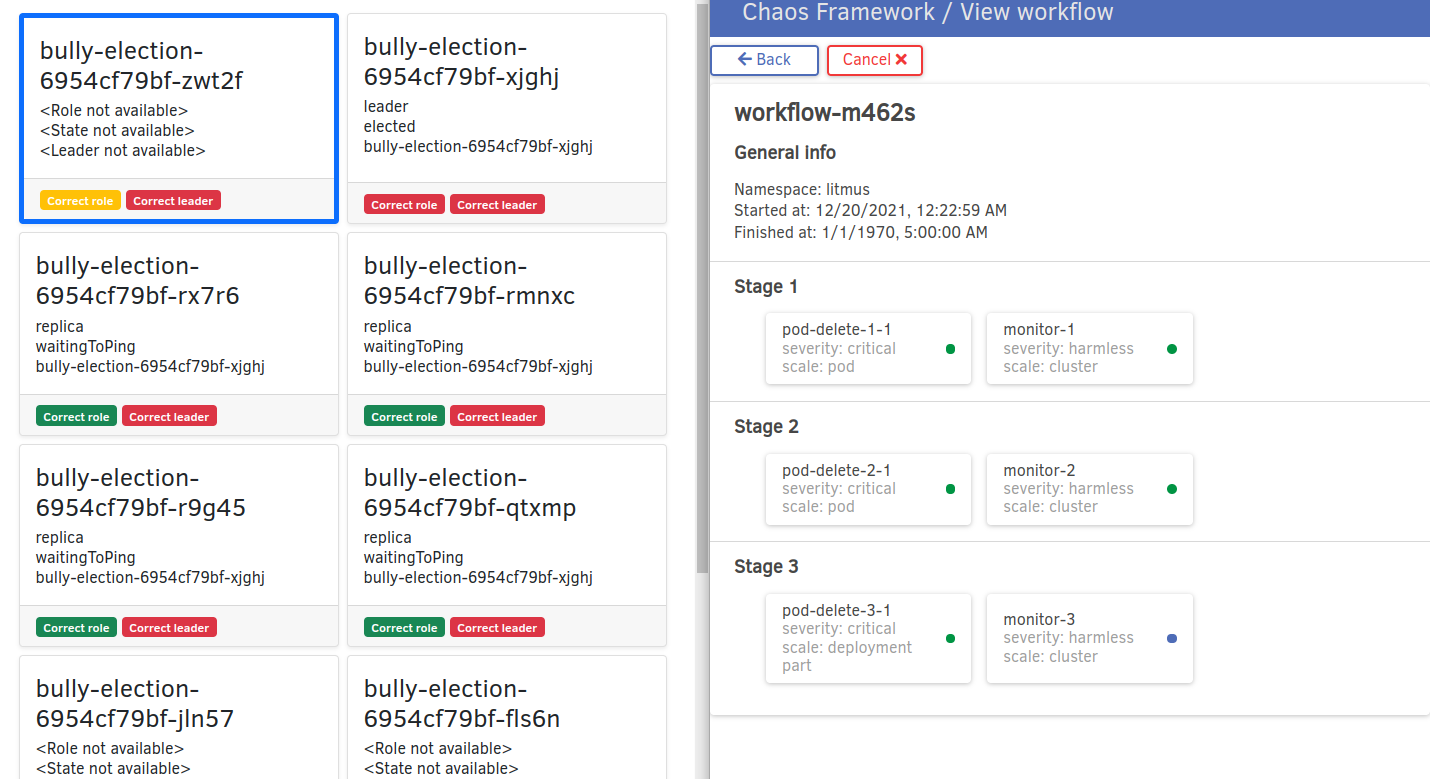


Нажмите на кнопку New. В данный момент поддерживается запуск тестов только в пространстве имен chaos-app. Вы можете выбрать seed, используемый для случайного выбора неисправностей и целей для каждой стадии теста. Также вы можете настроить состав и длительность теста: количество стадий из одной неисправности, количество стадий с однотипными неисправностями и количество стадий с разнообразными неисправностями



Оставьте выделенными только последнюю категорию неисправностей (restart). Нажмите Run





# Требования к тестируемой системе

Для корректной работы Chaos Framework должны выполняться следующие требования:

* должны использоваться только развертывания и сервисы (т. е. на данный момент нет поддержки DaemonSet, ReplicaSet, StatefulSet, Job, CronJob и т. п.). Создание подов напрямую тоже не поддерживается
* развертывания и поды ОБА должны иметь совпадающие метки (labels) вида “app: <name>” (см. пример с NGINX: там “app: nginx”)
* тестируемая система должна быть запущена в пространстве имен chaos-app

# Дополнения

* Вы можете использовать Argo Workflows для просмотра состояния теста, т. к. именно Argo Workflows используется для выполнения тестов (см. развертывание argo-server в пространстве имен argo)
* В каждую стадию теста запускается монитор, который следит за падениями компонентов тестируемой системы. К сожалению, в данный момент единственный способ узнать название упавшего компонента - посмотреть логи монитора через Argo Workflows
* Указывайте у используемых образов (особенно у ваших приложений) версию в качестве тега, т. к. по умолчанию при “imagePullPolicy: IfNotPresent” Kubernetes не будет проверять, обновился ли ваш образ myimage с предыдущего запуска. Напротив, если вы указываете версии в качестве тега (myimage:v1, myimage:v2 и т. д.), то Kubernetes при изменении образа в развертывании с myimage:v1 на myimage:v2 загрузит новую версию и пересоздаст поды уже с ней
* У неисправностей в скобочках указан масштаб (под/часть развертывания (по умолчанию 50%)/развертывание/нода) и их серьезность (для сетевых - потеря/повреждение/дублирование 10%/90% пакетов или 3с/9с для сетевой задержки соответственно для легкой и серьезной неисправности)