



Kubernetes and Service Monitoring with Prometheus

Hand-on

- Setup Prometheus Monitoring on Kubernetes Cluster
- Slack Notifications & Grafana
- Horizontal Pod Autoscale with Custom Prometheus Metrics

Prometheus ?

Monitoring system & time series database

Prometheus ?

- Prometheus는 원래 SoundCloud 에서 제작 된 시스템 모니터링 및 알람 툴
- 독립형 오픈 소스 프로젝트이며 어떤 회사와도 독립적으로 유지 관리
- 모든 구성 요소는 Apache 2 License이며 GitHub에서 사용가능
- Cloud Native Computing Foundation

Prometheus 주 기능

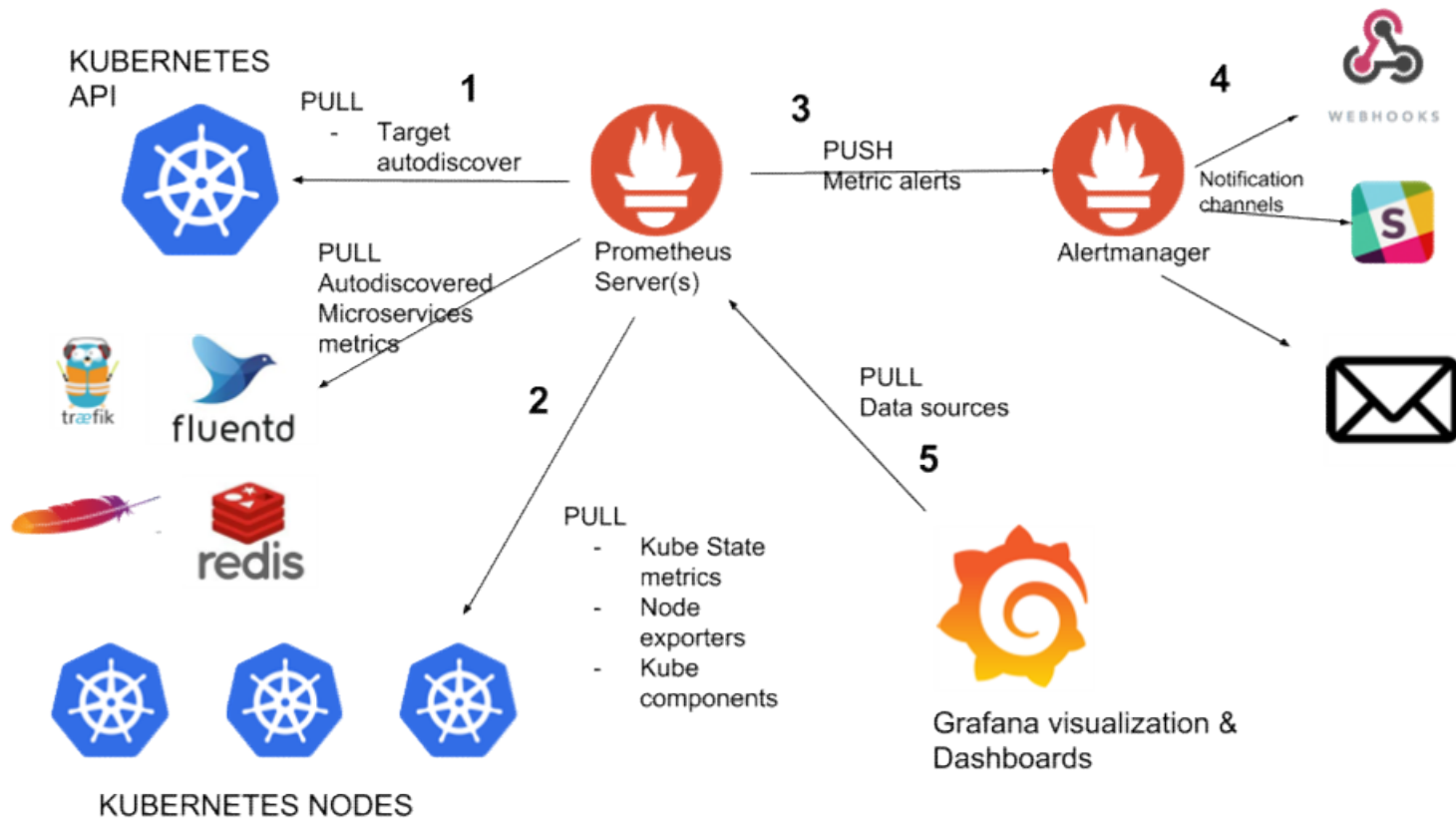
- 다차원 데이터 모델 메트릭 이름과 키/값 쌍으로 식별된 시계열 데이터가 포함
- PromQL을 활용하는 유연한 쿼리 언어
- 분산 스토리지에 의존하지 않음, 단일 서버 노드가 독립적임
- Time series 수집은 HTTP를 통한 풀 모델을 통해 수행
- 다중 그래핑 및 대시보드 지원 모드

Prometheus 구성요소

- Prometheus server - 시계열 데이터를 수집하고 저장
- 응용 프로그램 코드 계측을 위한 클라이언트 라이브러리
- Push gateway - batch job 등 서비스 수준 배치 작업의 결과를 캡처
- HAProxy, StatsD, Graphite 등과 같은 서비스를 위한 exporters
- 알람들을 처리하는 alertmanager

프로메테우스는 여러 가지 요소로 이루어져 있으며, 그 중 다수는 선택사항이다.

Prometheus Architecture



Prometheus는 직접 메트릭을 수집하거나 Pushgateway를 통해 메트릭을 스크랩한다.

스크랩 된 모든 샘플을 로컬에 저장하고 이 데이터에 대한 규칙을 실행하여 데이터를 집계 및 기록하거나 알람을 생성한다.

Grafana 또는 다른 API 사용자는 수집 된 데이터를 시각화하는 데 사용할 수 있다.

When does it fit?

When does it fit?

- 프로메테우스는 어떤 순수한 숫자 시계열이라도 기록하는 데 효과적
- 고도의 다이나믹 서비스 지향 아키텍처 모니터링뿐만 아니라 머신 중심 모니터링에도 적합
- Prometheus는 장애 발생시 신속하게 문제를 진단 할 수 있도록 안정성을 위해 설계
- Prometheus 서버는 네트워크 저장소나 다른 원격 서비스에 의존하지 않는 독립형 서버
- 인프라의 다른 부분이 고장 났을 때 의지 할 수 있으며 이를 사용하려면 광범위한 인프라를 구축이 필요없다

When does it not fit?

- Prometheus는 Prometheus server의 신뢰성을 중요시 한다
- Billing 시스템과 같이 100% 정확도가 필요하고 상세한 수집데이터가 필요한 경우 적합하지 않다
- 서비스문제가 발생했을 때 이를 검색해서 어떤 일이 있었는지의 원인을 밝히고자 했을 때는 적합하지 않다

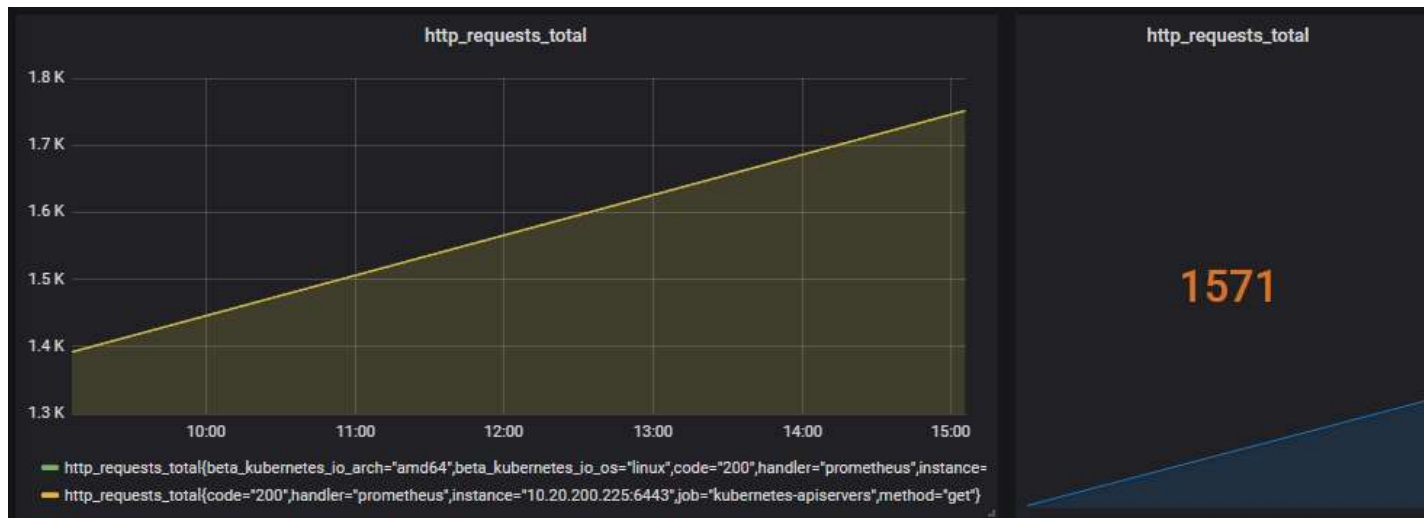
METRIC TYPES

Prometheus 클라이언트 라이브러리에서 제공하는 핵심 메트릭 유형

● Counter

카운터는 단일 나타내는 누적 메트릭이며 증가하는 카운터 값에만 사용가능하며 재시작을 0으로 재설정 한다
카운터를 사용하여 제공된 요청, 완료된 작업 또는 오류 수를 나타낼 수 있다

http total request등을 측정 할 때 사용한다



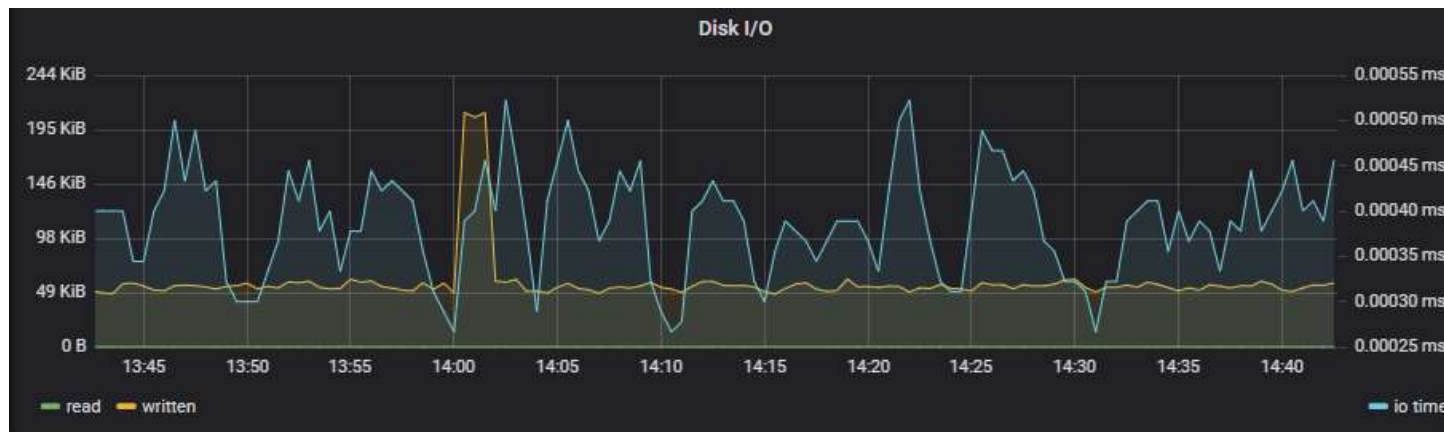
METRIC TYPES

Prometheus 클라이언트 라이브러리에서 제공하는 핵심 메트릭 유형

- Gauge

게이지를 임의로 위아래로 갈 수 있는 하나의 숫자 값을 보여주는 통계

게이지는 일반적으로 온도 나 현재 메모리 사용량과 같은 측정 값에 사용



METRIC TYPES

Prometheus 클라이언트 라이브러리에서 제공하는 핵심 메트릭 유형

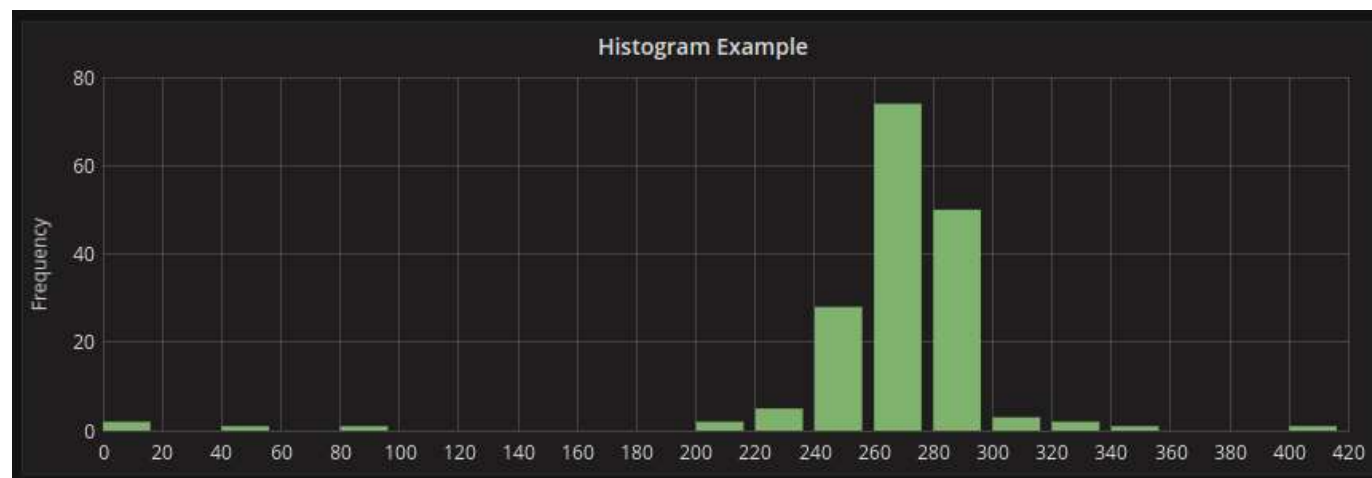
● Histogram

히스토그램은 숫자 데이터의 분포를 그래픽으로 표현한 것

샘플 값을 그룹화 한 다음 각 버킷에 얼마나 많은 값이 들어가는지 계산한다.

실제 값을 그래프로 표시하는 대신 버킷을 그래프로 표시합니다.

각 막대는 버킷을 나타내고 막대 높이는 해당 버킷의 간격에 속한 값의 빈도 (즉, 개수)를 나타냅니다.



Installing kubernetes

실습환경

OS

- CentOS 7.6

CPU/Memory/Disk

- 2 CPU
- 8 GB
- Disk 90GB

Kubernetes 방화벽

Master node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443*	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10251	kube-scheduler	Self
TCP	Inbound	10252	kube-controller-manager	Self

Worker node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	30000-32767	NodePort Services**	All

Installing kubernetes

- SeLinux disable

```
$ setenforce 0  
$ sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

- Firewalld disable

```
$ systemctl stop firewalld  
$ systemctl disable firewalld
```

- 설치 스크립트 경로 이동

```
$ cd /root/kubernetes-accordion-course/k8s-install  
$ ls  
0.Prerequisites 1.docker 2.kubernetes 3.helm
```

Installing kubernetes

- Docker 설치 및 실행

```
# 필수 패키지 설치
$ yum install -y yum-utils device-mapper-persistent-data lvm2

# docker repo 추가
$ yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

# docker-ce 18.06.3버전 설치
$ yum install -y --setopt=obsoletes=0 docker-ce-18.06.3.ce-3.el7.x86_64

# docker 시작
$ systemctl enable docker
$ systemctl restart docker

or

$ cd /root/kubernetes-accordion-course/k8s-install/1.docker
$ ./1.docker_install.sh
```


Installing kubernetes

- Kubernetes Repo 추가

```
$ /root/kubernetes-accordion-course/k8s-install/2.kubernetes/1.add_k8s_repo.sh
$ cat /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kube*
```

- Kubernetes 설치

```
$ yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
$ systemctl enable -now

or
$ /root/kubernetes-accordion-course/k8s-install/2.kubernetes/2.k8s_install.sh
```

Installing kubernetes

- Kubeadm init 실행

```
$ kubeadm init
[init] Using Kubernetes version: v1.15.0
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-
flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [mantech kubernetes kubernetes.default
[...]

Your Kubernetes master has initialized successfully!
```

Installing kubernetes

- Kube config 생성

```
# kube config 생성
$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
$ export KUBECONFIG=/etc/kubernetes/admin.conf

# pod network add-on 설치
$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr
-d '\n')"
```

마스터 노드 POD 배포 설정

```
$ kubectl taint nodes --all node-role.kubernetes.io/master-
```

or

```
$ /root/kubernetes-accordion-course/k8s-install/2.kubernetes/4.after_init.sh
```

Installing kubeadm, kubelet and kubectl

- Helm 바이너리 다운로드

```
$ wget https://get.helm.sh/helm-v2.14.1-linux-amd64.tar.gz
--2019-07-17 11:01:03-- https://get.helm.sh/helm-v2.14.1-linux-amd64.tar.gz
Resolving get.helm.sh (get.helm.sh)... 152.199.39.106, 2606:2800:247:b40:171d:1a2f:2077:f6b
Connecting to get.helm.sh (get.helm.sh)|152.199.39.106|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26532029 (25M) [application/x-tar]
Saving to: 'helm-v2.14.1-linux-amd64.tar.gz'
100%[=====>] 26,532,029
103MB/s in 0.2s
2019-07-17 11:01:04 (103 MB/s) - 'helm-v2.14.1-linux-amd64.tar.gz' saved [26532029/26532029]

$ tar zvxf helm-v2.14.1-linux-amd64.tar.gz
$ \cp -fp ./linux-amd64/helm /usr/bin/helm
$ helm
The Kubernetes package manager

or

$ /root/kubernetes-accordion-course/k8s-install/3.helm/1.helm_download.sh
```

Installing kubeadm, kubelet and kubectl

- Helm 초기화

```
$ kubectl create serviceaccount --namespace kube-system tiller
serviceaccount/tiller created

$ kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --
serviceaccount=kube-system:tiller
clusterrolebinding.rbac.authorization.k8s.io/tiller-cluster-rule created

$ helm init --service-account tiller
Creating /root/.helm
Creating /root/.helm/repository
...
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /root/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

or

$ /root/kubernetes-accordion-course/k8s-install/3.helm/2.helm_install.sh
```

Prometheus

Prometheus 설치

- Prometheus 설치

```
$ helm install stable/prometheus --name prometheus --namespace monitoring --set  
server.persistentVolume.enabled=false --set alertmanager.persistentVolume.enabled=false --set  
alertmanager.service.type=NodePort --set server.service.type=NodePort
```

- Prometheus DNS name, NodePort 확인

Kubernetes Cluster에서 사용할 수 있는 Prometheus-server의 DNS 이름
prometheus-server.monitoring.svc.cluster.local

```
$ kubectl get --namespace monitoring -o jsonpath="{.spec.ports[0].nodePort}" services prometheus-  
server  
32638
```

```
$ kubectl get --namespace monitoring -o jsonpath="{.spec.ports[0].nodePort}" services prometheus-  
alertmanager  
30094
```

Prometheus 설치

- Grafana 설치

```
$ helm install stable/grafana --name grafana --namespace monitoring --set service.type=NodePort --set adminPassword=admin
```

- Grafana NodePort 확인

Kubernetes Cluster에서 사용할 수 있는 grafana의 DNS 이름
grafana.monitoring.svc.cluster.local

```
$ kubectl get --namespace monitoring -o jsonpath="{.spec.ports[0].nodePort}" services grafana
30011
```

- Grafana URL에 접속하기 위해서는 웹브라우저에서 **http://공인IP:NodePort**를 입력하여 접속 합니다.

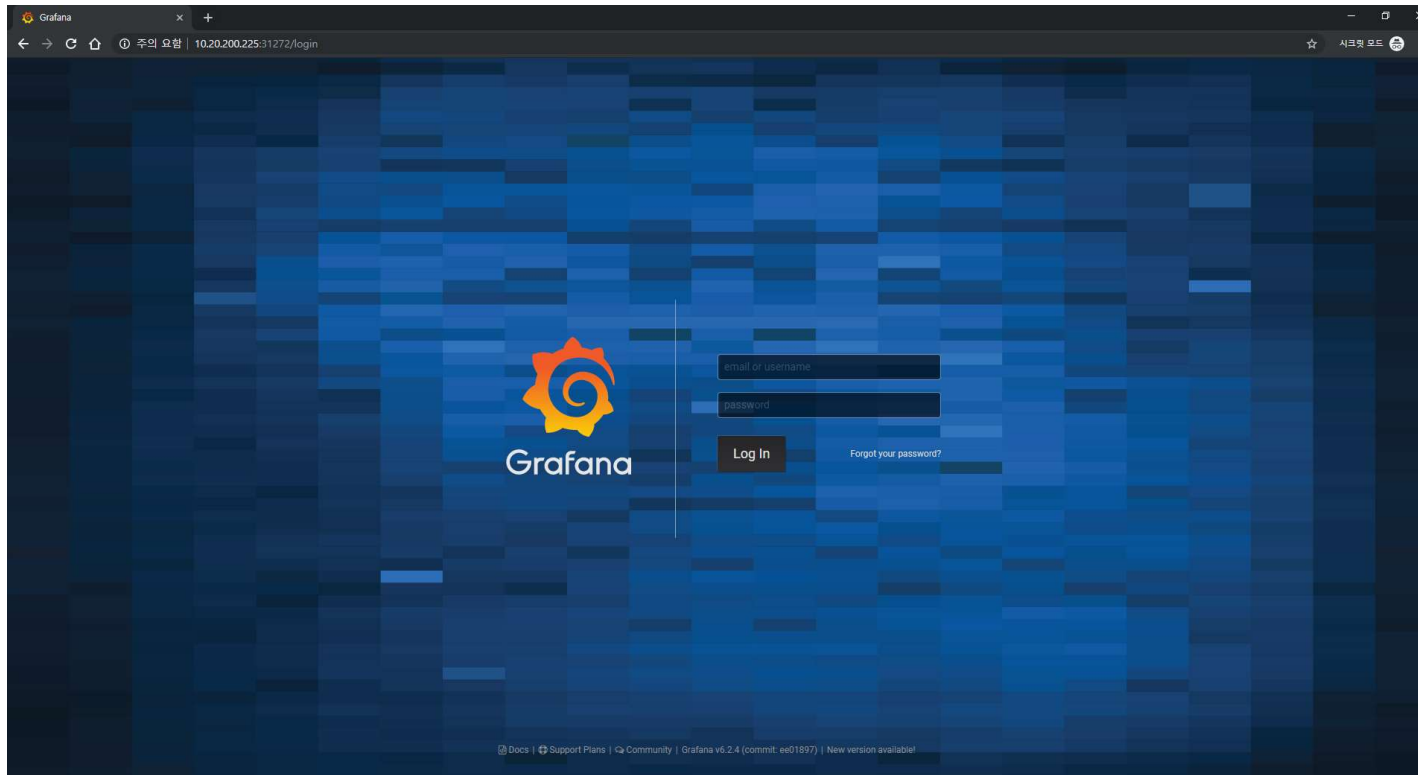
Grafana

Prometheus ?

- Prometheus는 원래 SoundCloud 에서 제작 된 시스템 모니터링 및 알람 툴
- 독립형 오픈 소스 프로젝트이며 어떤 회사와도 독립적으로 유지 관리
- 모든 구성 요소는 Apache 2 License이며 GitHub에서 사용가능
- Cloud Native Computing Foundation

Grafana 사용하기

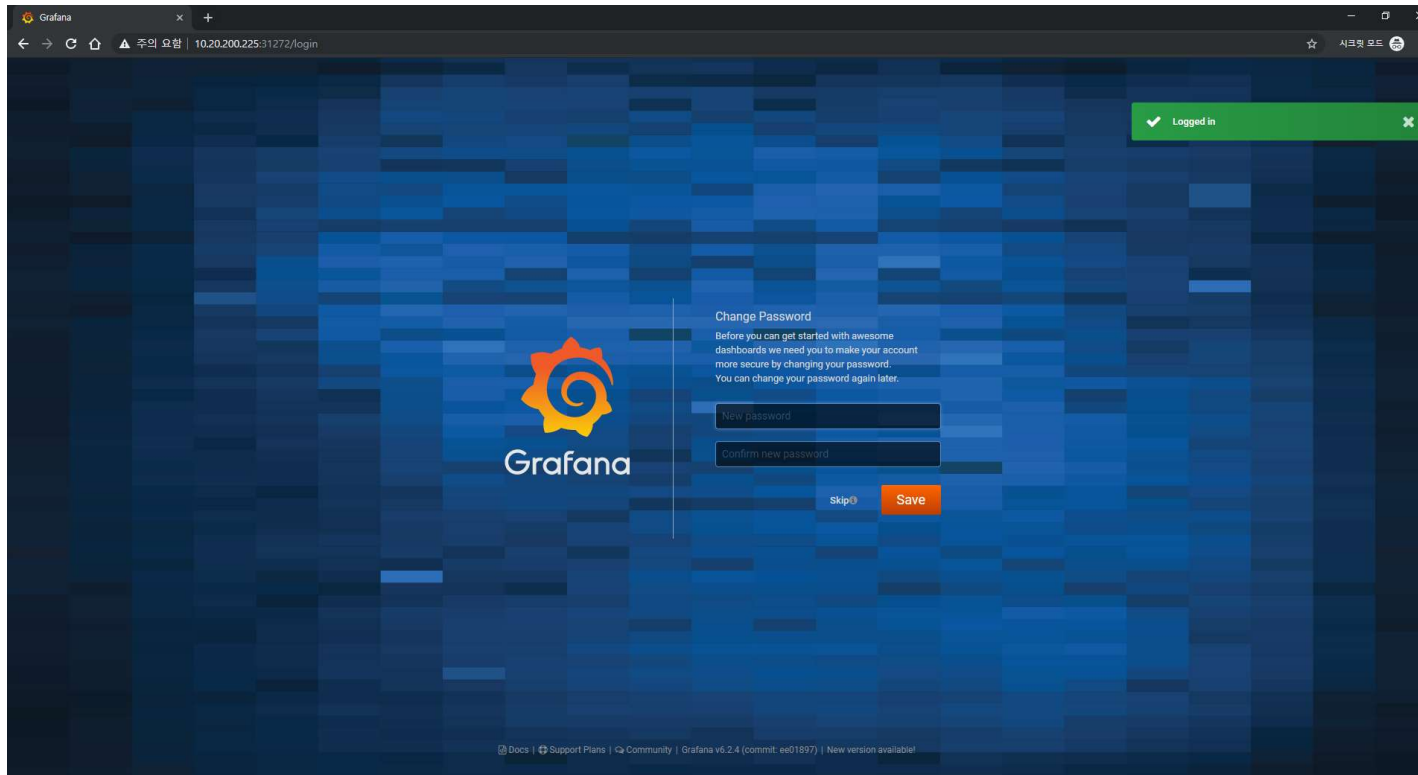
- Grafana URL 접속



- 웹브라우저에서 `http://공인IP:NodePort`를 입력하여 Grafana URL에 접속 합니다.
- Username: admin
- Password: admin

Grafana 사용하기

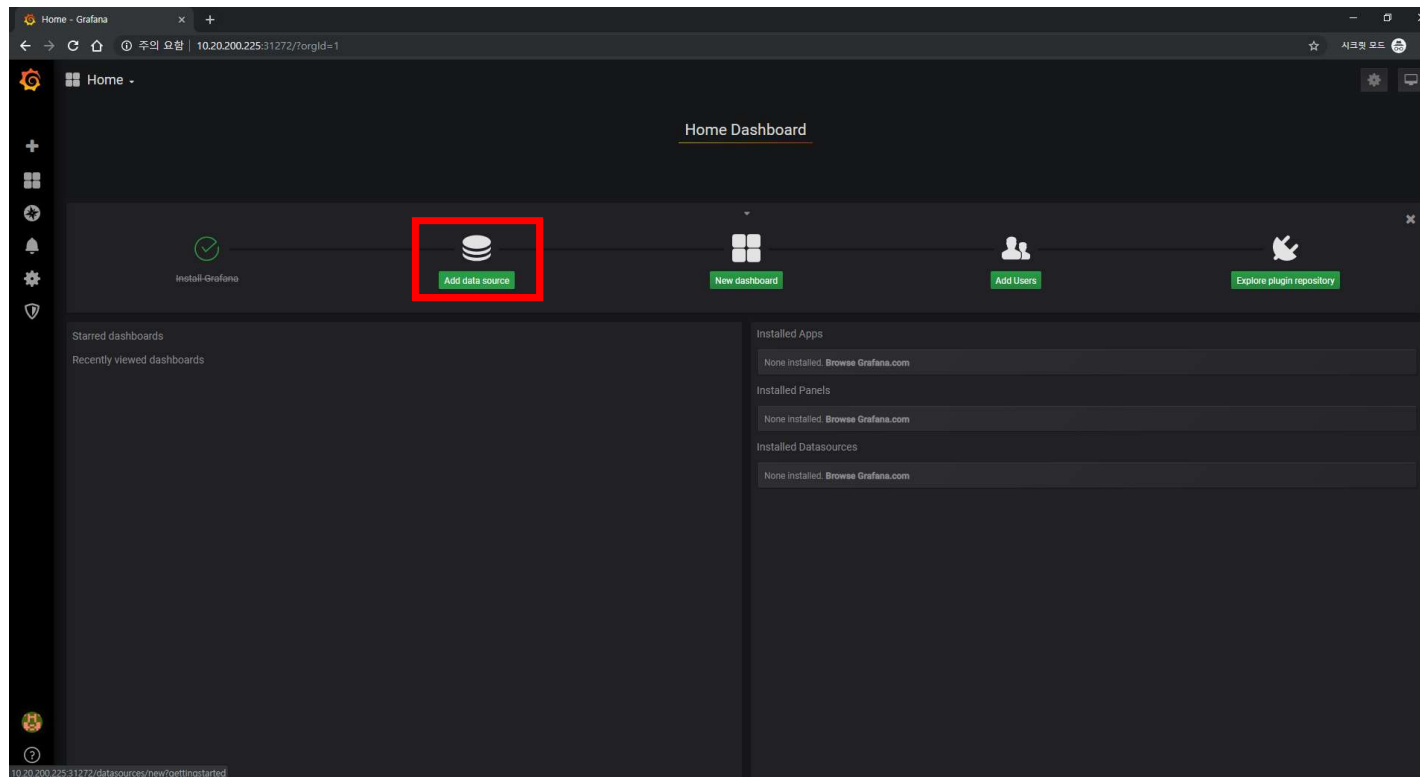
- Grafana 로그인



- 단순한 패스워드 사용으로 로그인 시 새로운 패스워드로 변경 합니다.

Grafana 사용하기

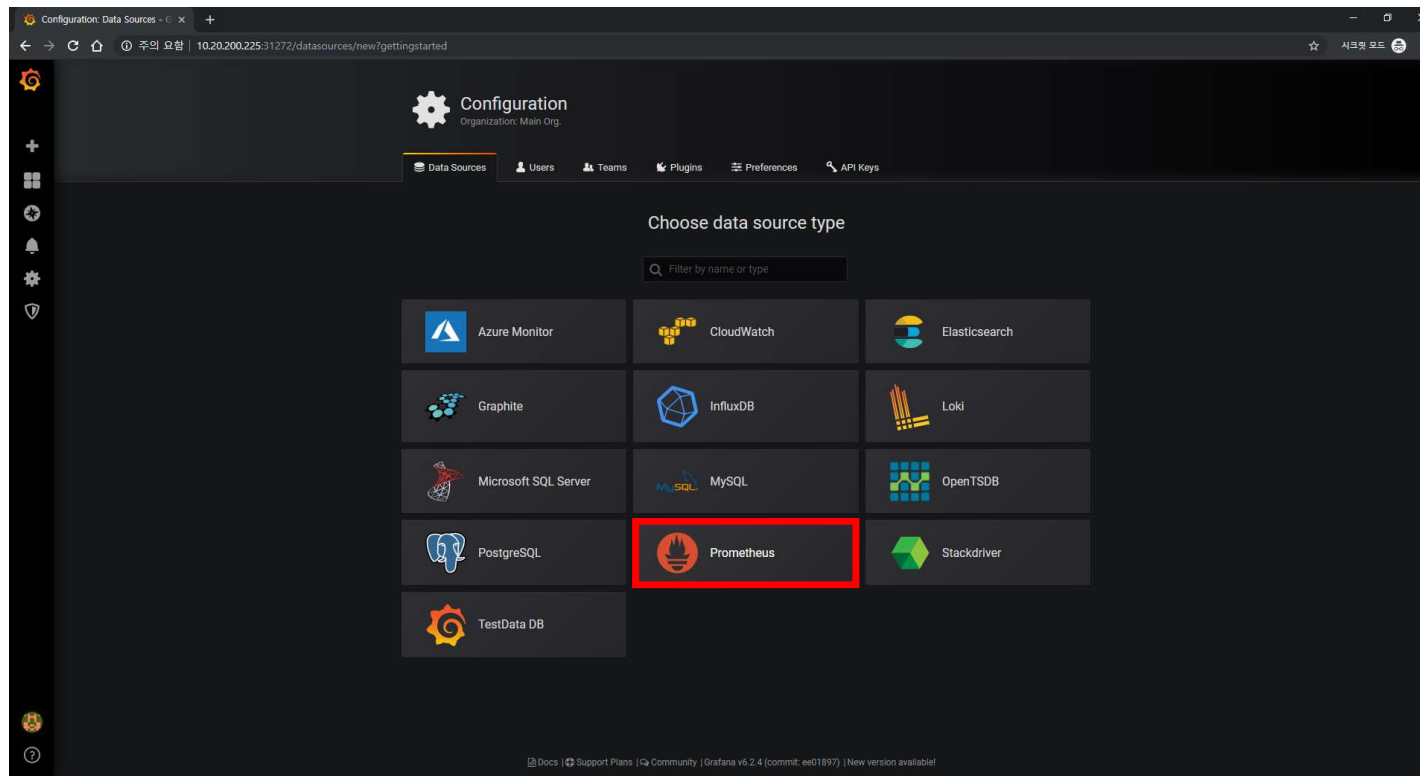
- Data source 추가(1/4)



- Prometheus 연동을 위해 Add data source를 클릭 합니다.

Grafana 사용하기

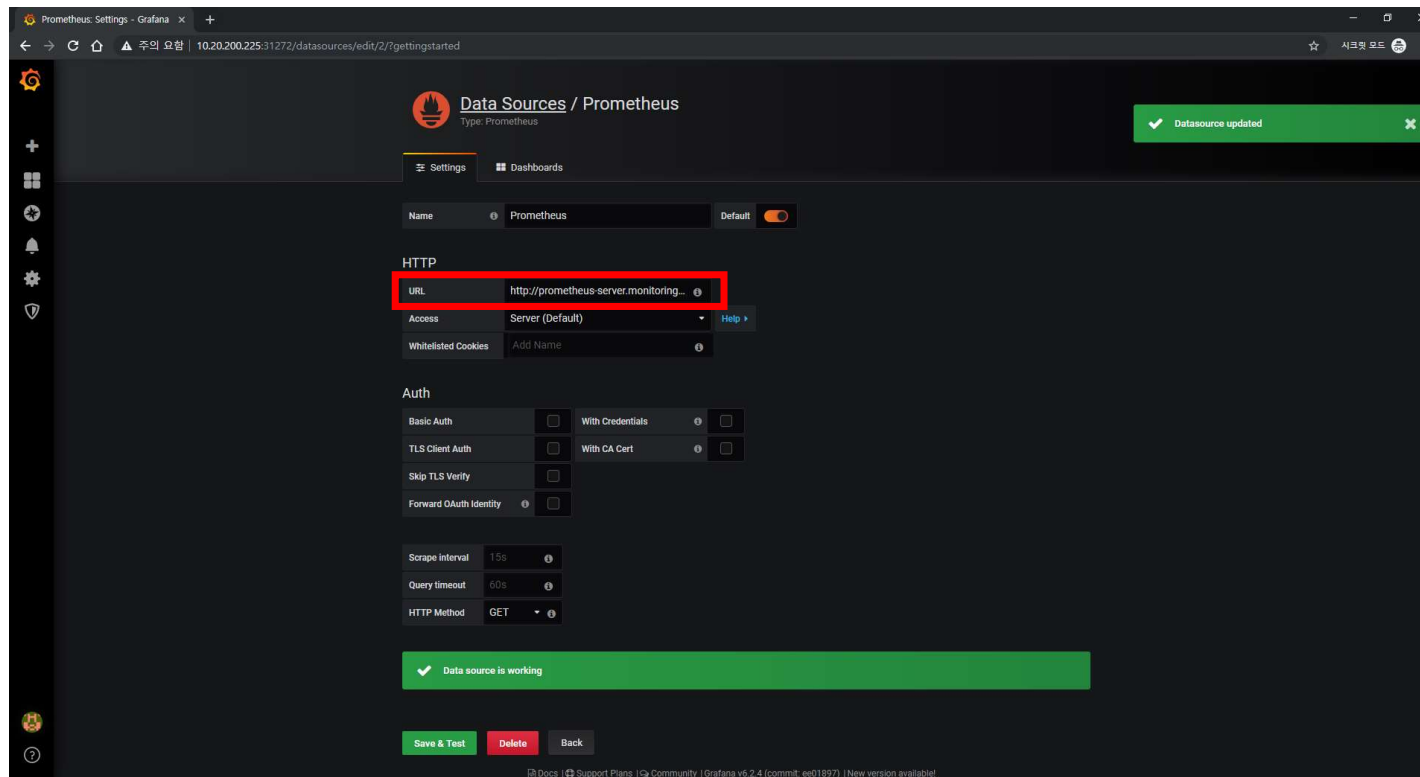
- Data source 추가(2/4)



- 중앙 하단에 Prometheus를 클릭 합니다.

Grafana 사용하기

- Data source 추가(3/4)

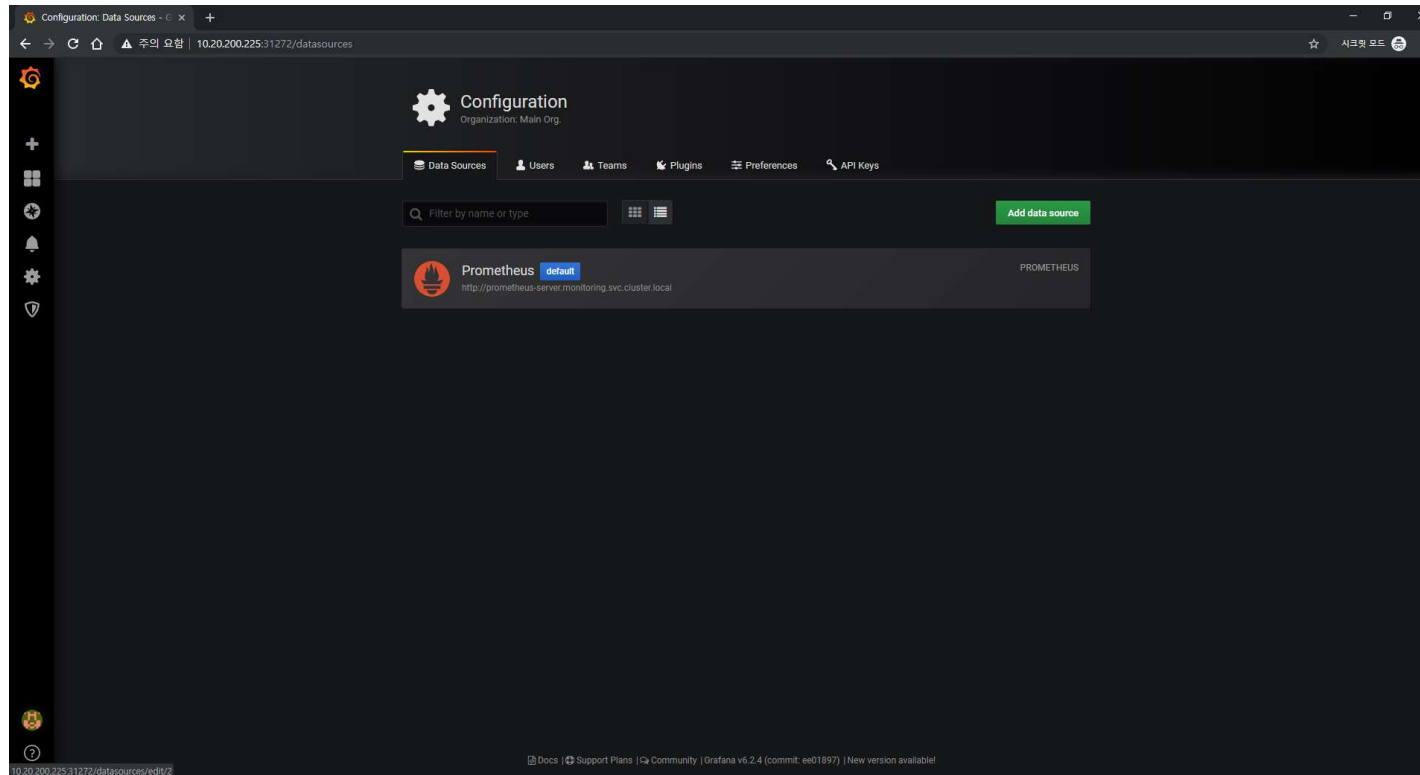


- HTTP URL을 입력 후 Save & Test를 클릭 합니다.

- HTTP URL: `http://prometheus-server.monitoring.svc.cluster.local`

Grafana 사용하기

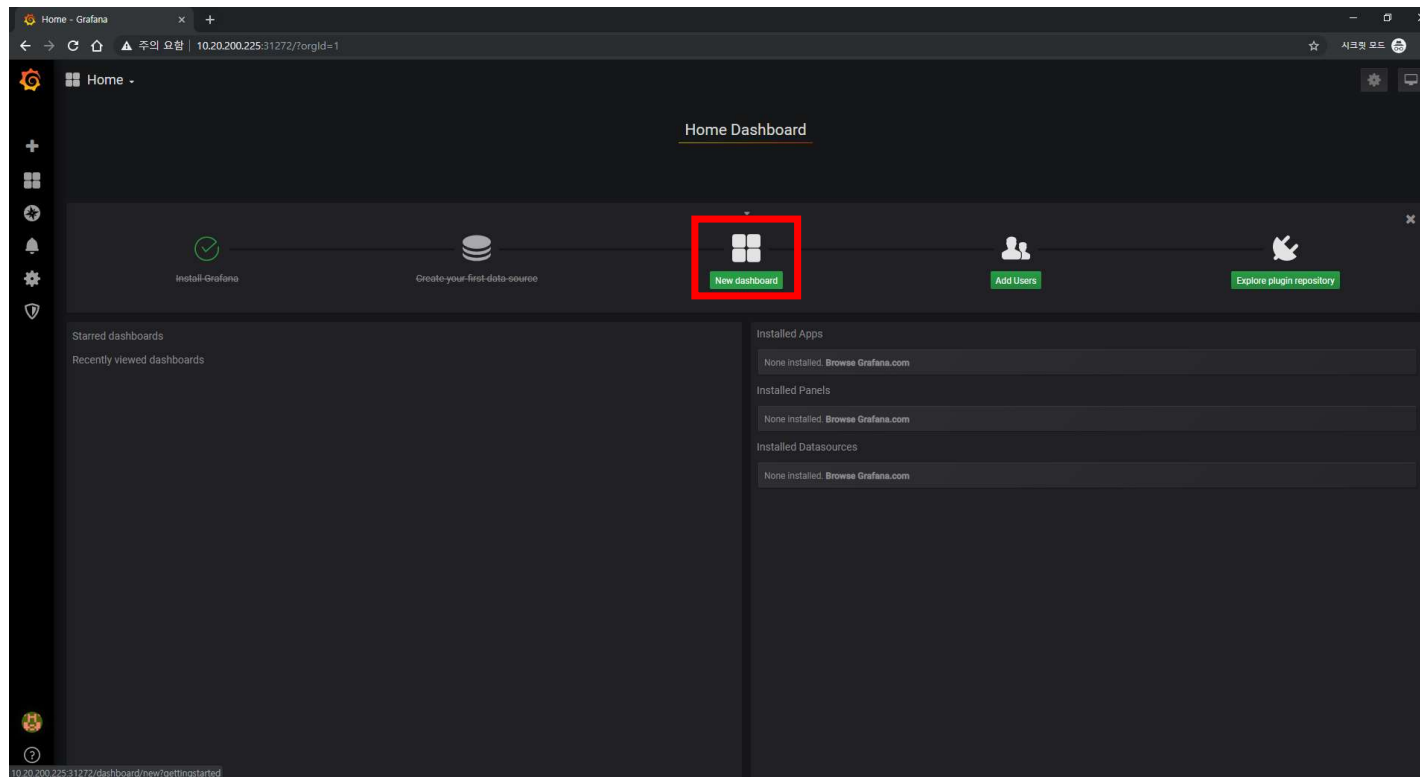
- Data source 추가(4/4)



- Back 버튼을 클릭하여 아래와 같은 화면이 나오면 Prometheus와 연동이 완료 되었습니다.

Grafana 사용하기

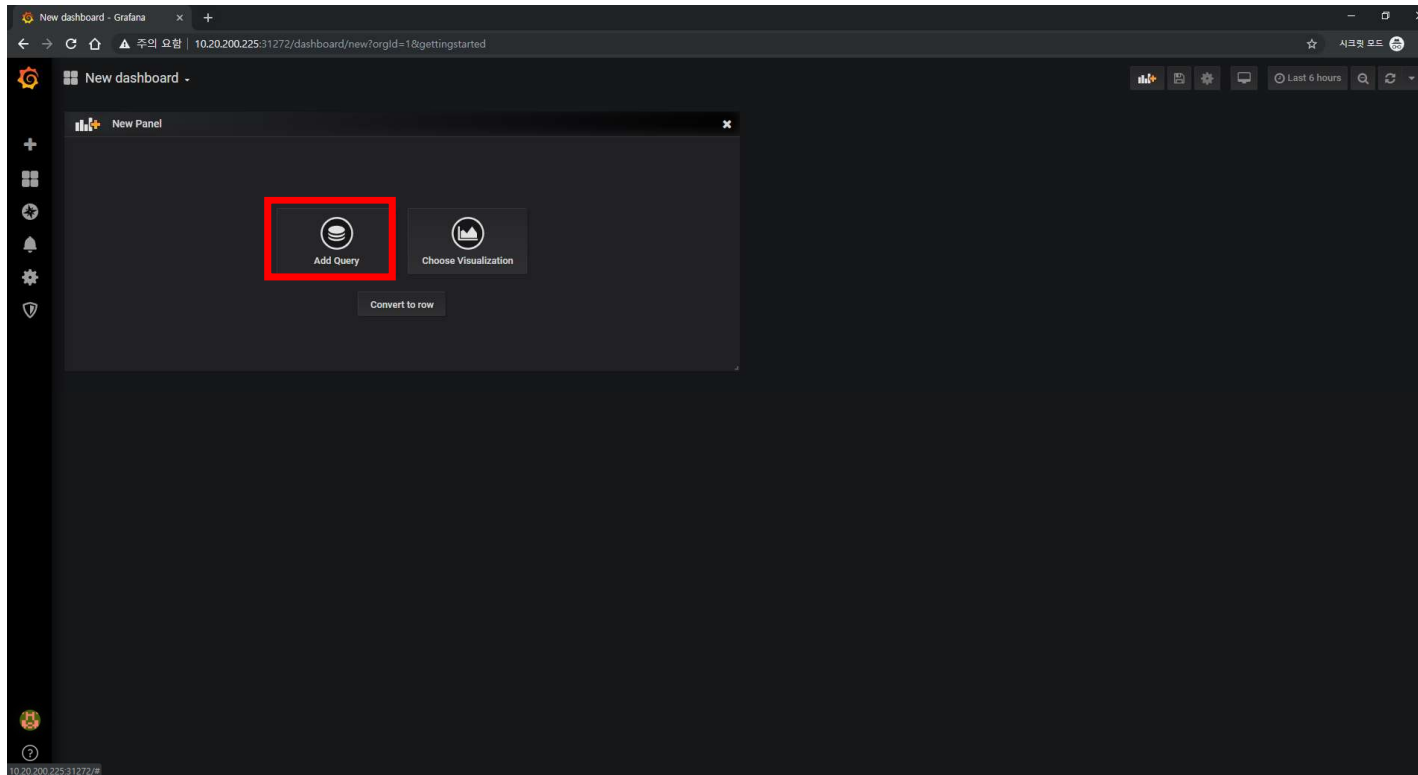
- Dashboard 생성 (1)



- Grafana의 Dashboard는 직접 생성 하거나 <https://grafana.com/dashboards> 에 공유된 Dashboard를 사용할 수도 있습니다.
- 모니터링을 위한 dashboard를 생성하기 위해 초기화면에서 New dashboard를 클릭합니다.

Grafana 사용하기

- Dashboard 생성 (2)



- CPU사용량을 확인하는 패널 생성을 위해 Add Query를 클릭 합니다.

Grafana 사용하기

- Dashboard 생성 (3)

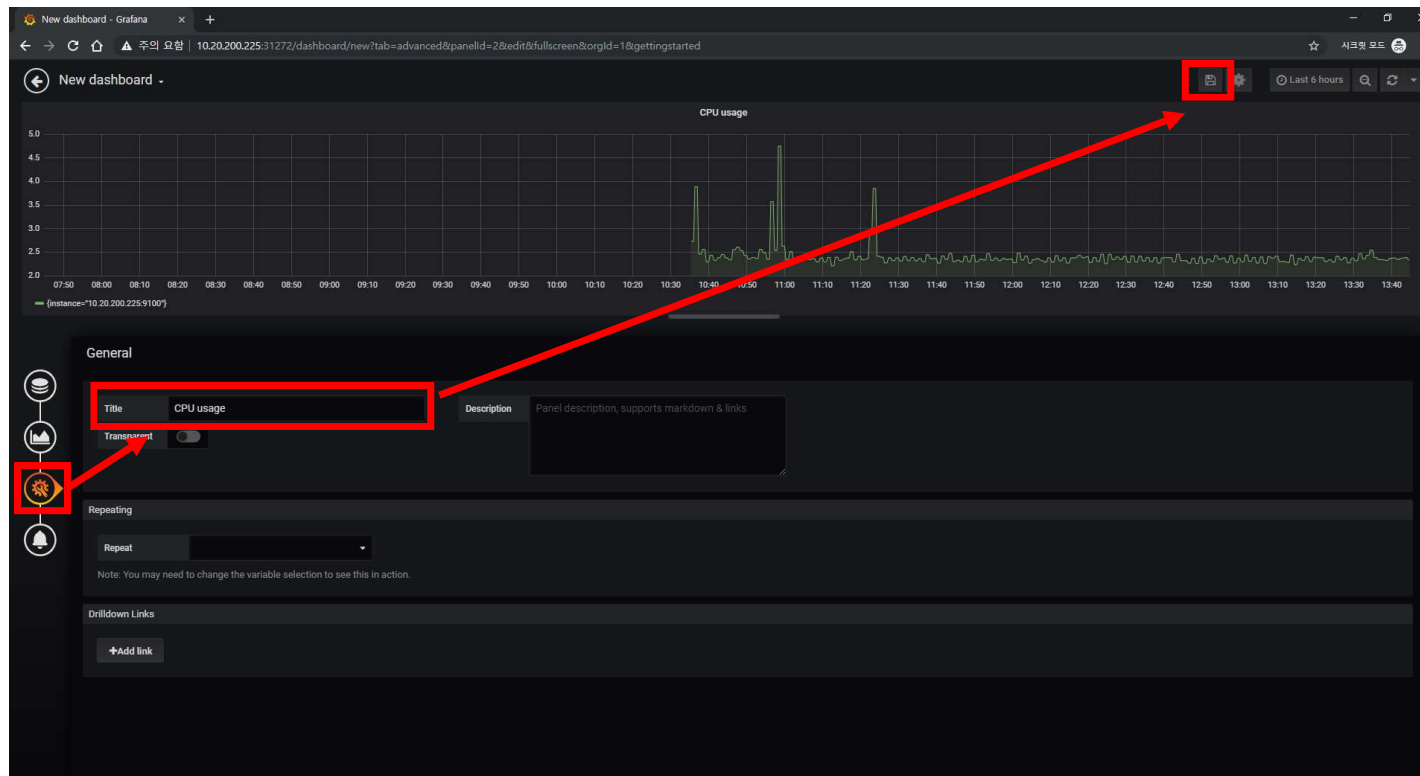


- 노드의 CPU사용량을 확인하는 쿼리를 입력 합니다.

- Query: `100- (avg by(instance) (irate(node_cpu_seconds_total{mode="idle"}[2m])) * 100)`

Grafana 사용하기

- Dashboard 생성 (4)



- 일반 설정에서 Title을 입력 후 Dashboard를 저장 합니다.

Grafana 사용하기

- Dashboard 생성 (5)



- 저장을 하고 나면 화면과 같이 CPU사용량을 모니터링하는 panel이 생성 됩니다.

Grafana 사용하기

- Dashboard 생성 (6)

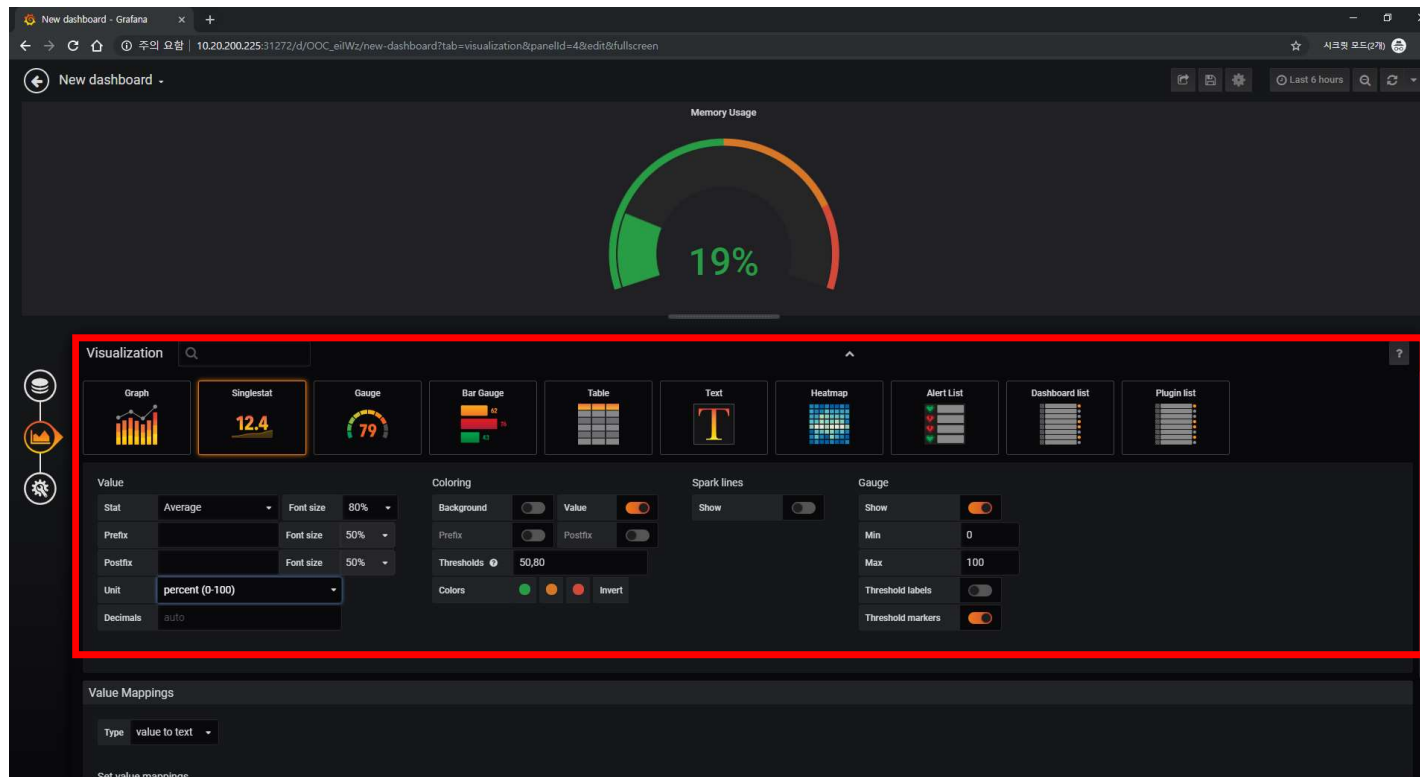


- 메모리 사용량 패널을 추가 합니다.

- Query: $(node_memory_MemTotal_bytes - node_memory_MemFree_bytes - node_memory_Buffers_bytes - node_memory_Cached_bytes) / node_memory_MemTotal_bytes * 100$

Grafana 사용하기

- Dashboard 생성 (7)

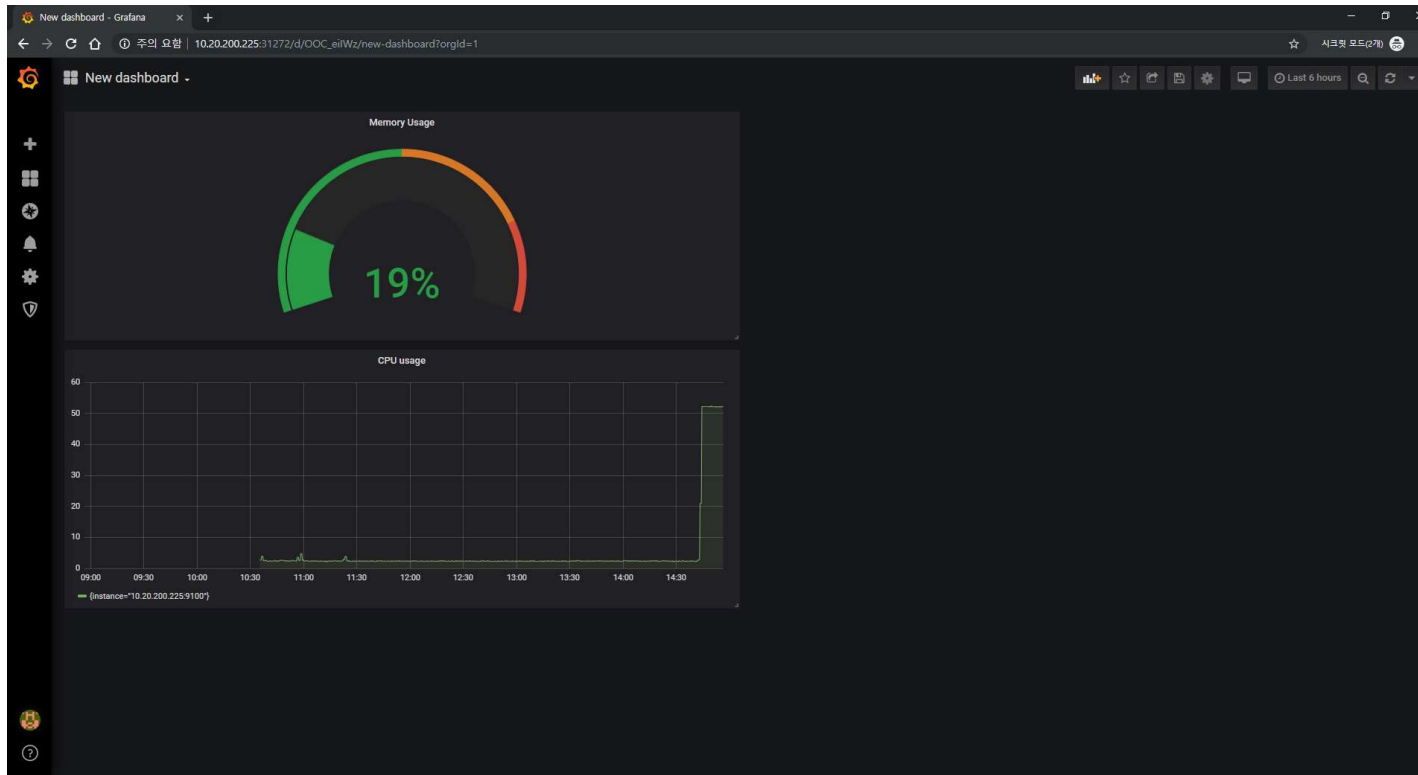


- 메모리 사용량은 Singlestat 형식으로 패널을 추가 합니다.

- Visualization: Singlestat / Value - Unit: percent (0-100) / Coloring - Value: true, Thresholds: 50,80 / Gauge - Show: true, Threshold markers: true

Grafana 사용하기

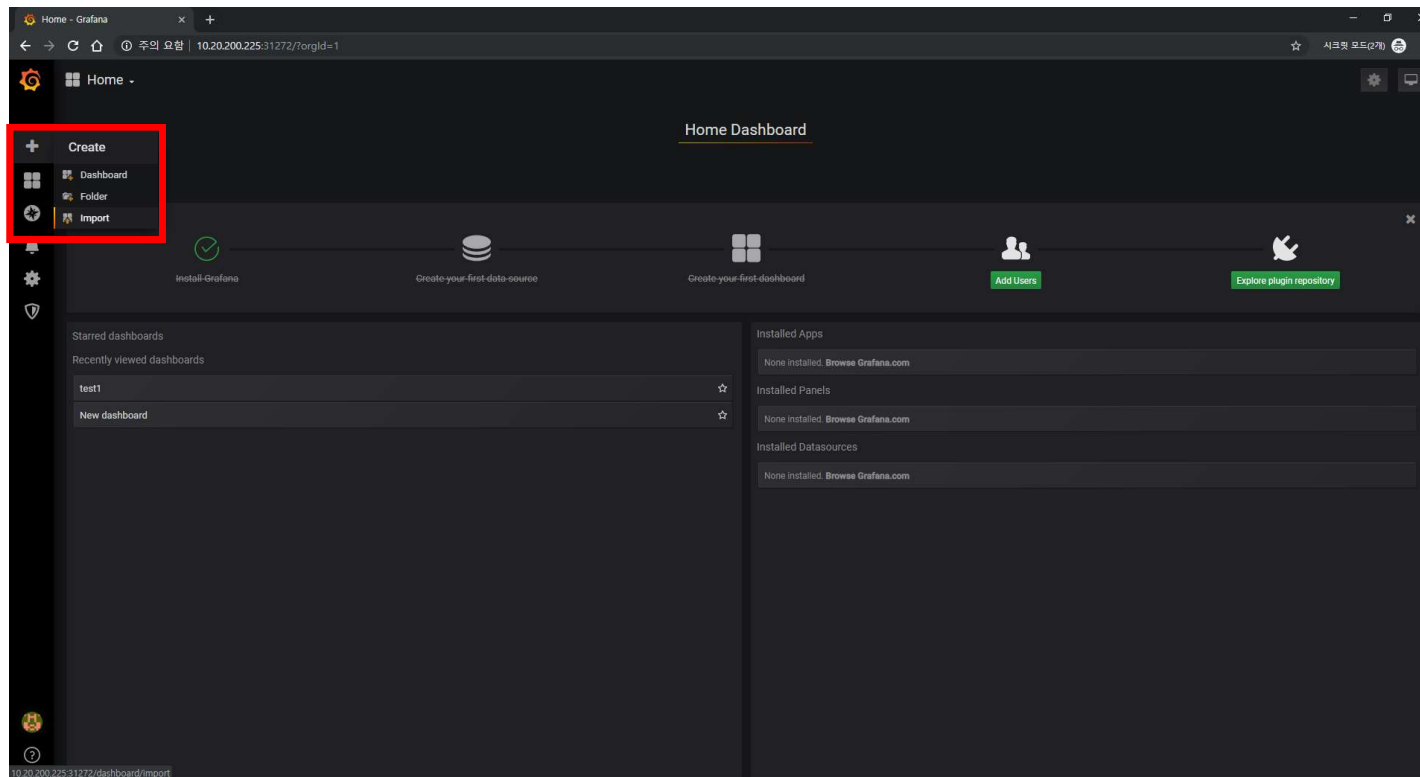
- Dashboard 생성 (8)



- Dashboard로 이동하면 생성한 메모리 사용량 패널이 생성되어 있습니다.

Grafana 사용하기

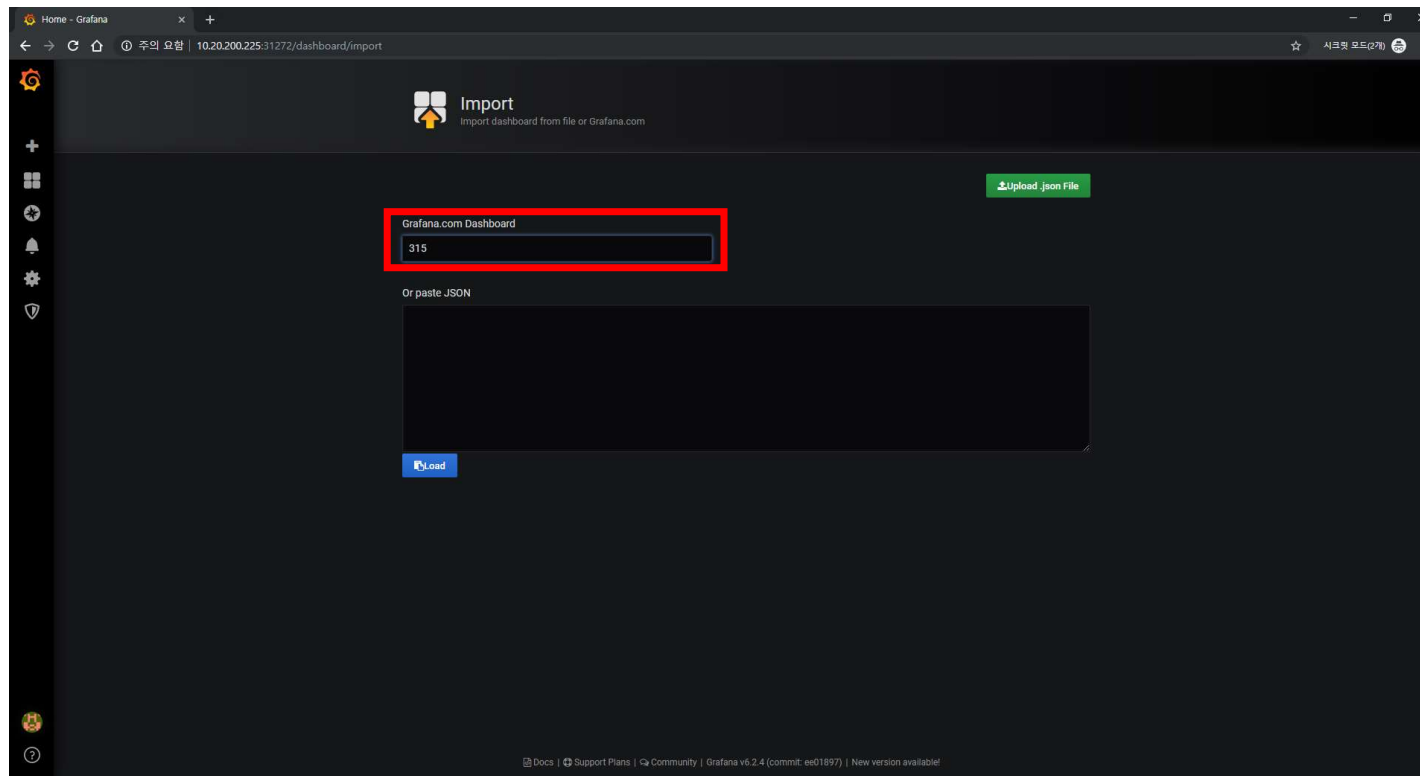
- 공유된 Dashboard Import하기 (1)



- Grafana 에서 Create → Import를 클릭 합니다.

Grafana 사용하기

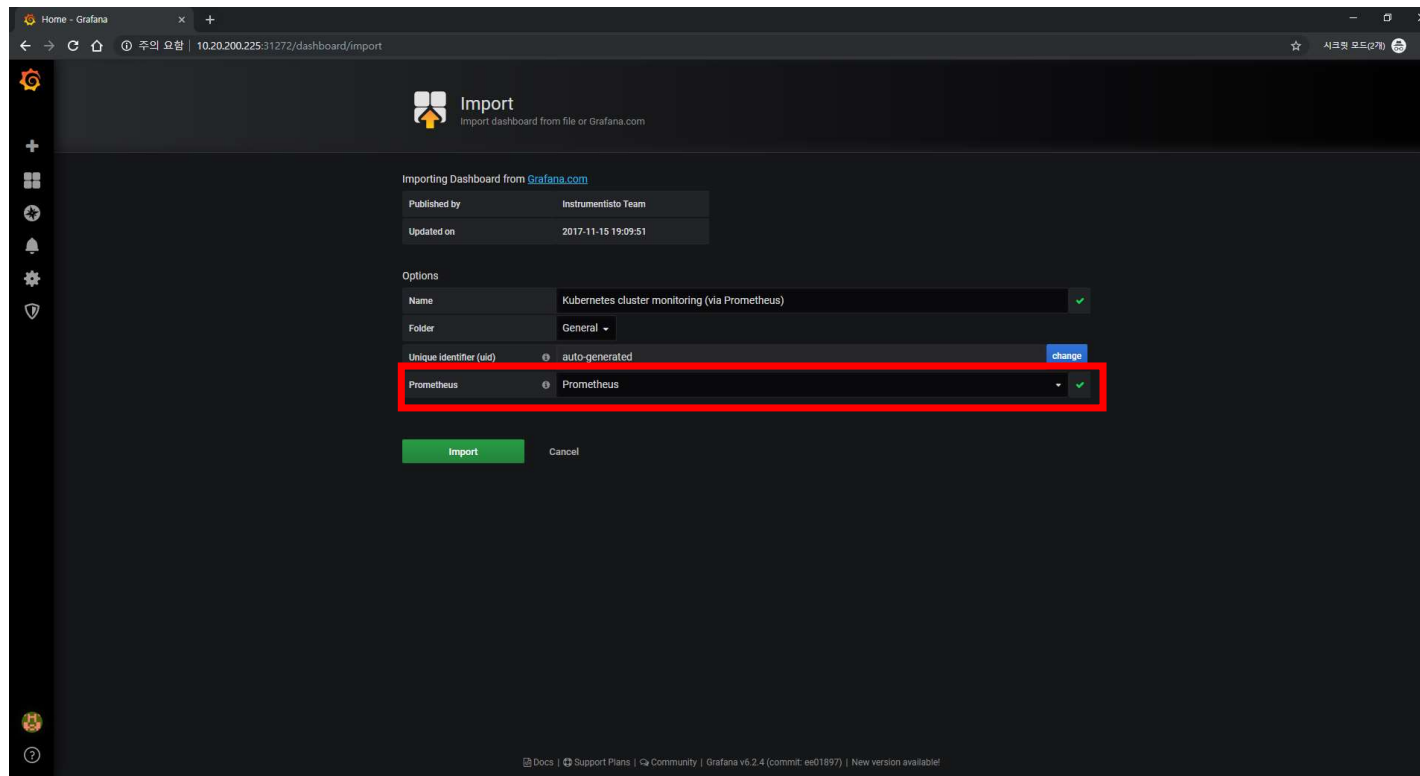
- 공유된 Dashboard Import하기 (2)



- <https://grafana.com/dashboards> 에 공유된 Dashboard ID 또는 json파일로 쉽게 Import할 수 있습니다.
- Dashboard 315번을 Import 합니다 315 또는 <https://grafana.com/dashboards/315> 를 입력 합니다

Grafana 사용하기

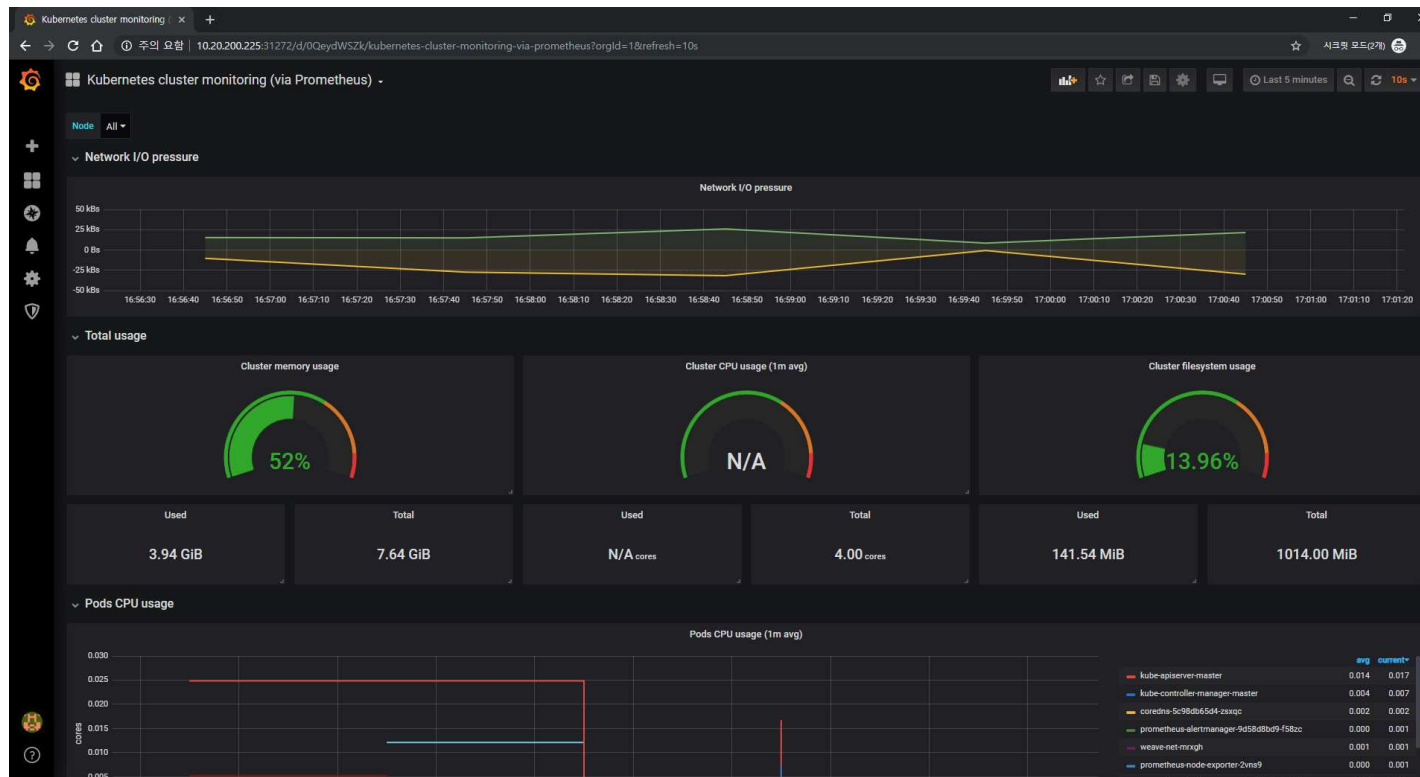
- 공유된 Dashboard Import하기 (3)



- Prometheus 데이터소스를 선택 후 Import 버튼을 클릭 합니다.

Grafana 사용하기

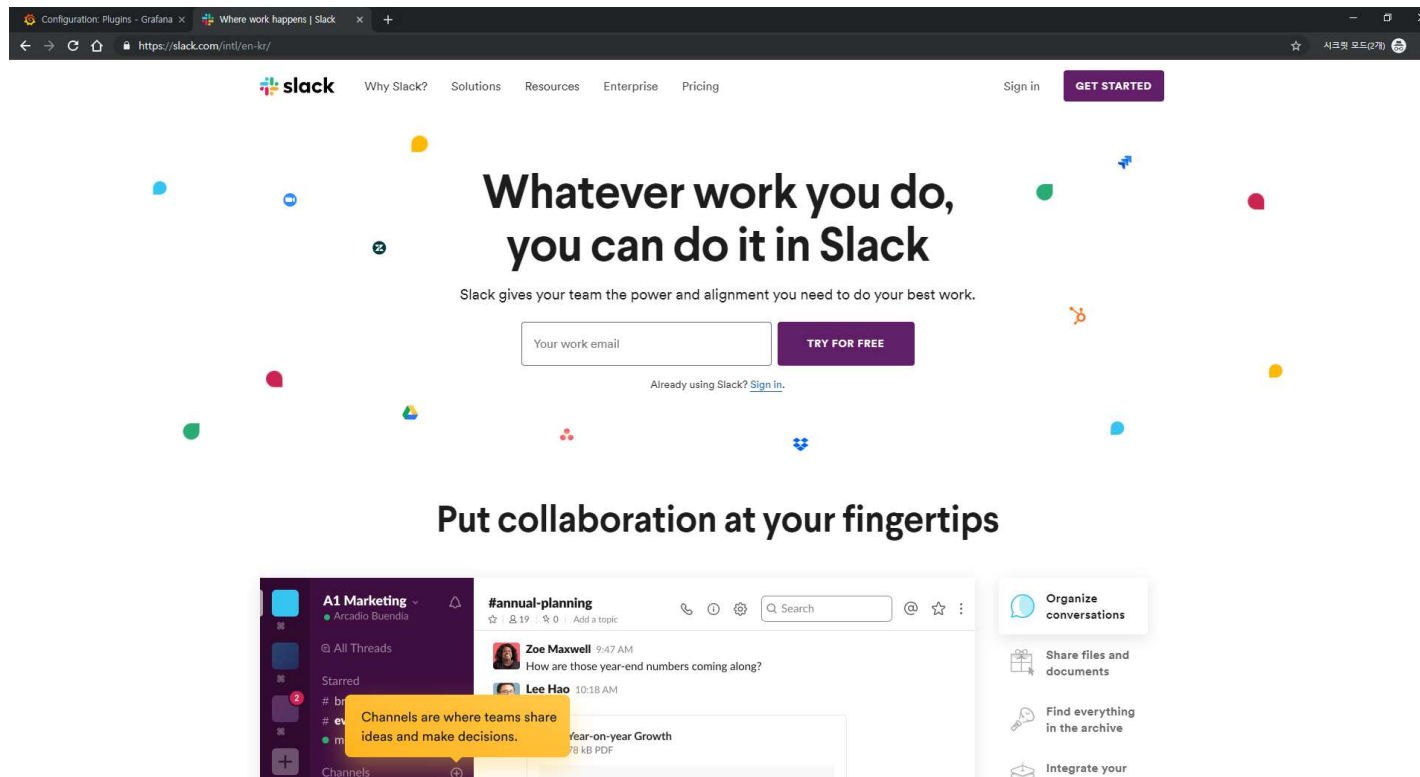
- 공유된 Dashboard Import하기 (4)



- Dashboard Import가 되면 자동으로 impor한 Dashboard로 이동 합니다.

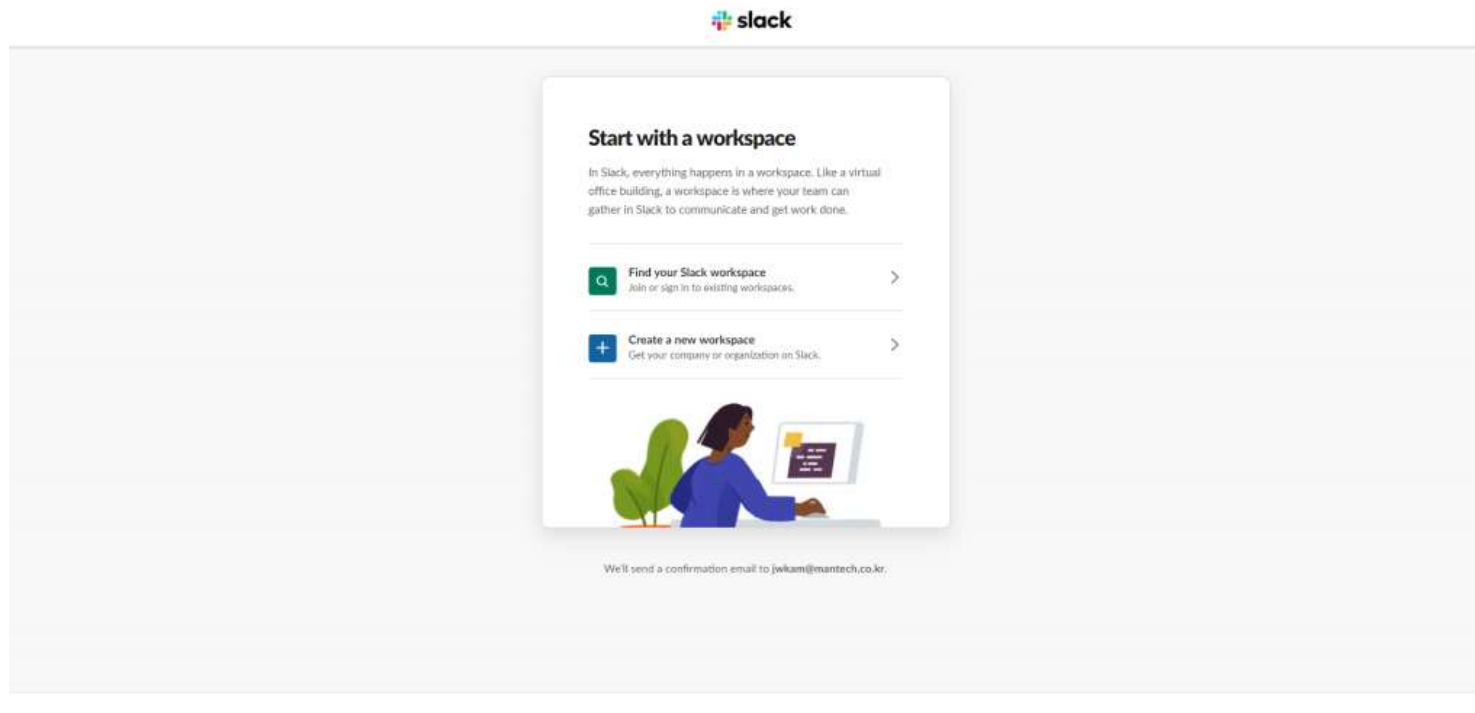
Grafana Alerting and Slack

- Slack 생성 (1)



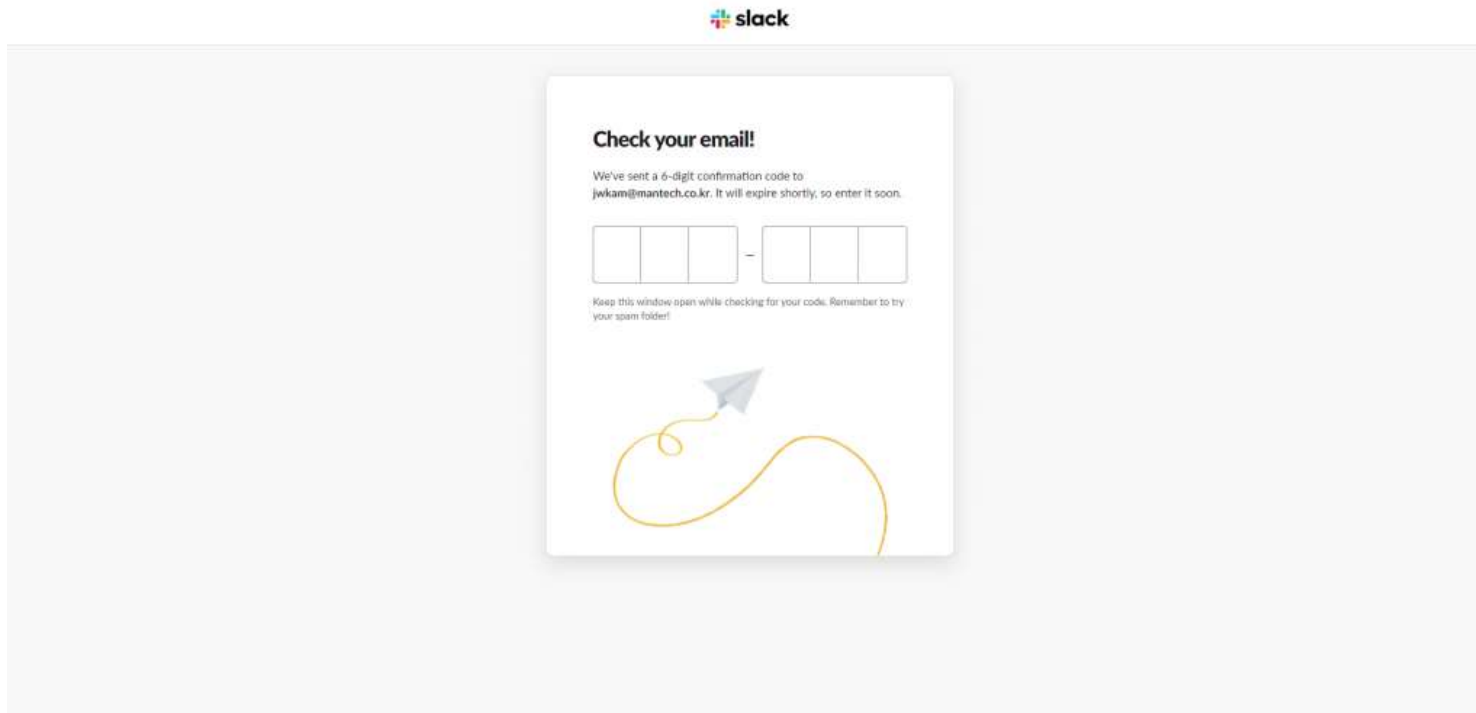
- prometheus alertmanager에서 slack으로 알람을 보내기 위해서는 "slack api url"과 "channel" 정보가 필요합니다.
- 브라우저에서 <https://slack.com> 에 접속 합니다.

- Slack 생성 (2)



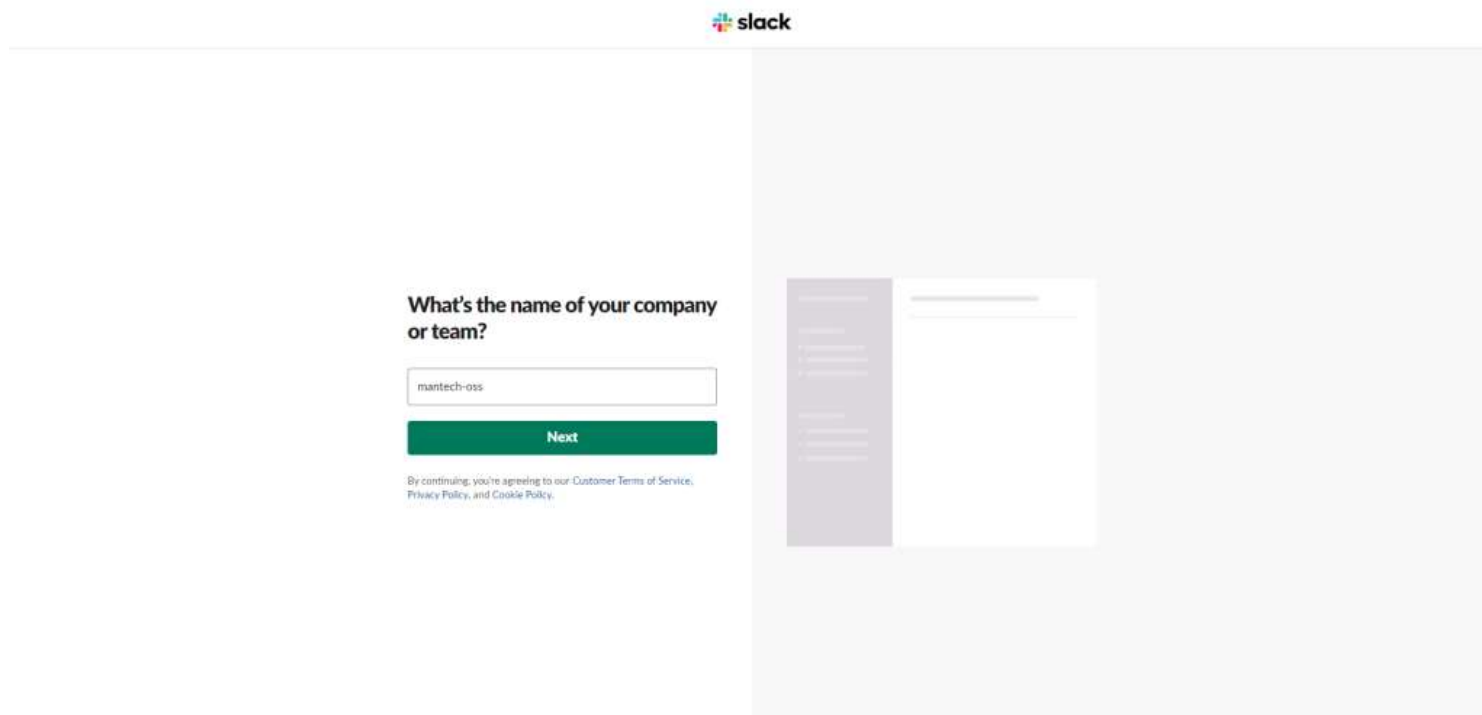
- Create a new workspace 를 클릭하여 workspace를 생성 합니다.

- Slack 생성 (3)



- 이메일 인증 창이 나옵니다 입력한 주소로 수신된 이메일에서 인증번호를 입력 합니다.

- Slack 생성 (4)

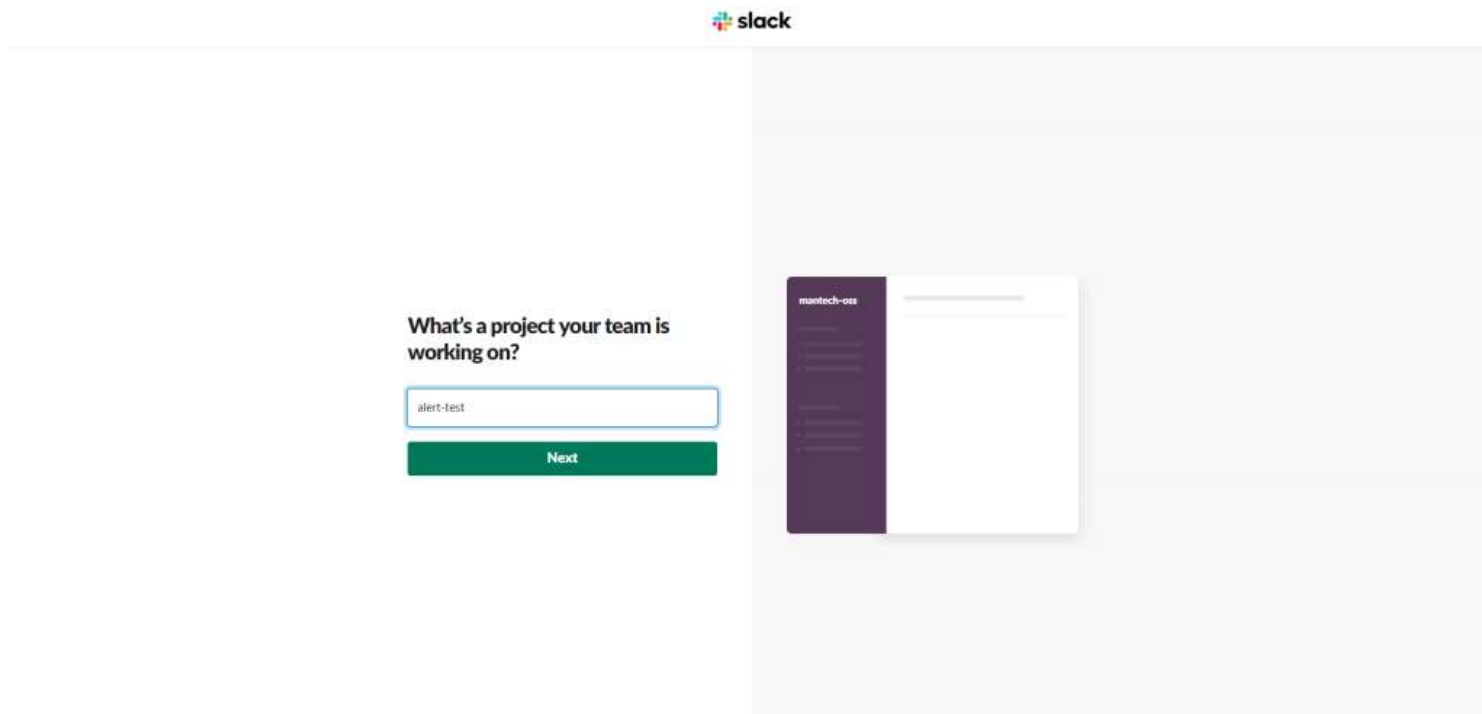


The image shows the Slack onboarding interface. At the top, the Slack logo is centered. Below it, the text "What's the name of your company or team?" is displayed. A text input field contains the text "mantech-oss". Below the input field is a green button labeled "Next". At the bottom, there is a small line of text: "By continuing, you're agreeing to our [Customer Terms of Service](#), [Privacy Policy](#), and [Cookie Policy](#)." To the right of the main form, there is a preview of the Slack interface, showing a sidebar with a list of channels and a main content area.

- 회사 또는 부서명을 입력 합니다.

Slack

- Slack 생성 (5)

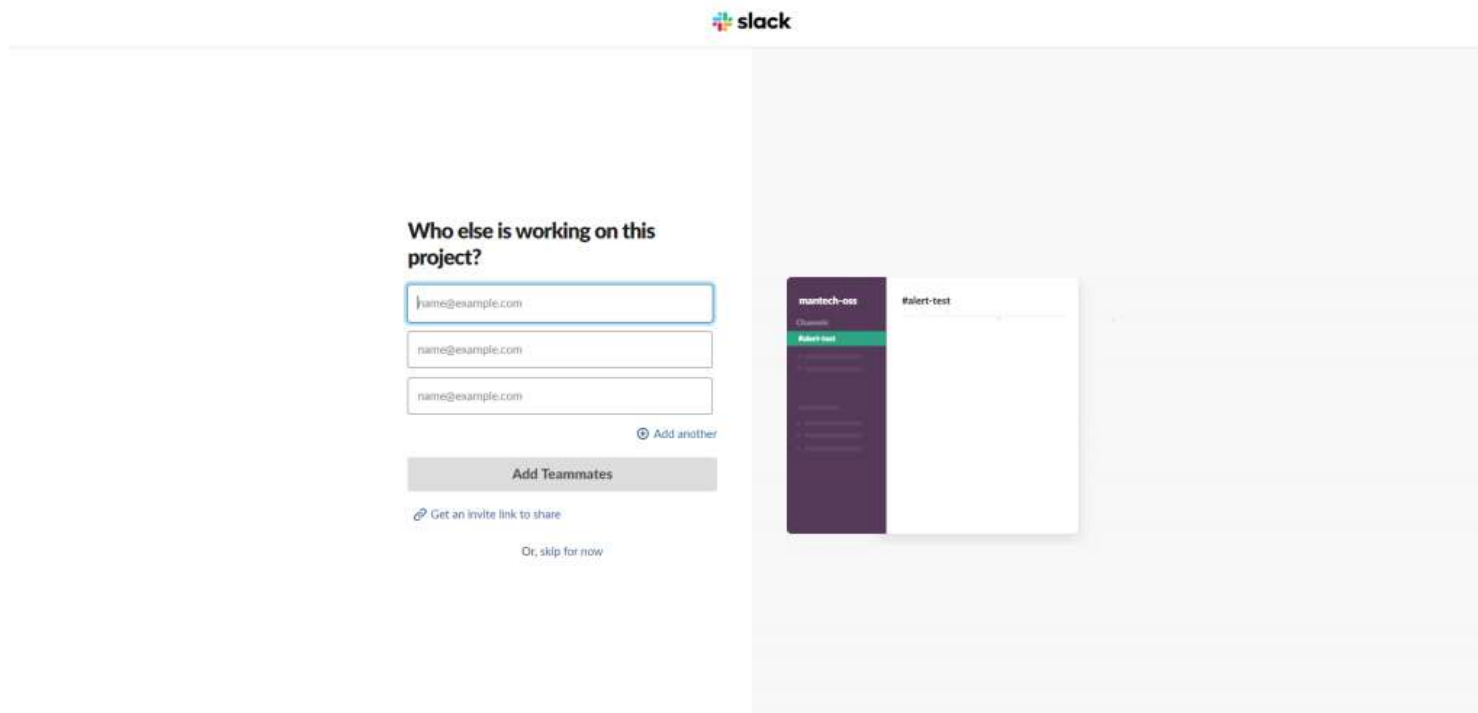


The image shows the Slack channel creation interface. At the top, the Slack logo is visible. The main heading asks, "What's a project your team is working on?". Below this is a text input field containing the text "alert-test". A green "Next" button is positioned below the input field. To the right of the input field, there is a preview of a Slack channel interface with a dark purple sidebar and a white main content area.

- 프로젝트(채널)명을 입력 합니다.

Slack

- Slack 생성 (6)

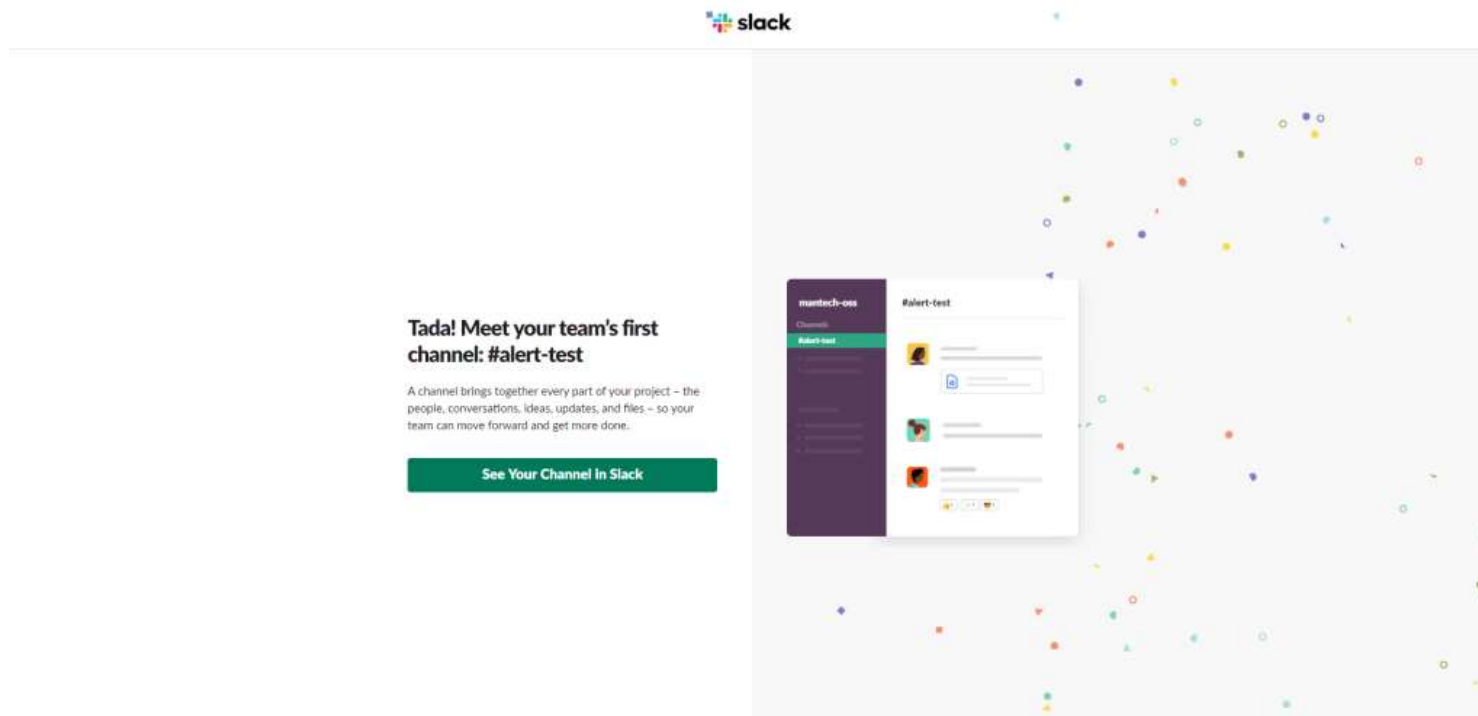


The image shows the Slack workspace creation interface. At the top, the Slack logo is visible. The main heading is "Who else is working on this project?". Below this, there are three input fields, each containing the placeholder text "name@example.com". To the right of the third input field is a link that says "Add another". Below the input fields is a grey button labeled "Add Teammates". Underneath the button is a link that says "Get an invite link to share". At the bottom, there is a link that says "Or, skip for now". On the right side of the screen, there is a preview of a Slack workspace. The preview shows a dark sidebar with the text "mantech-oss" and "#alert-test" highlighted. The main area of the preview shows a channel named "#alert-test" with a list of messages.

- 초대할 사람의 이메일 주소를 입력합니다. 없으면 "skip for now"를 클릭 합니다.

Slack

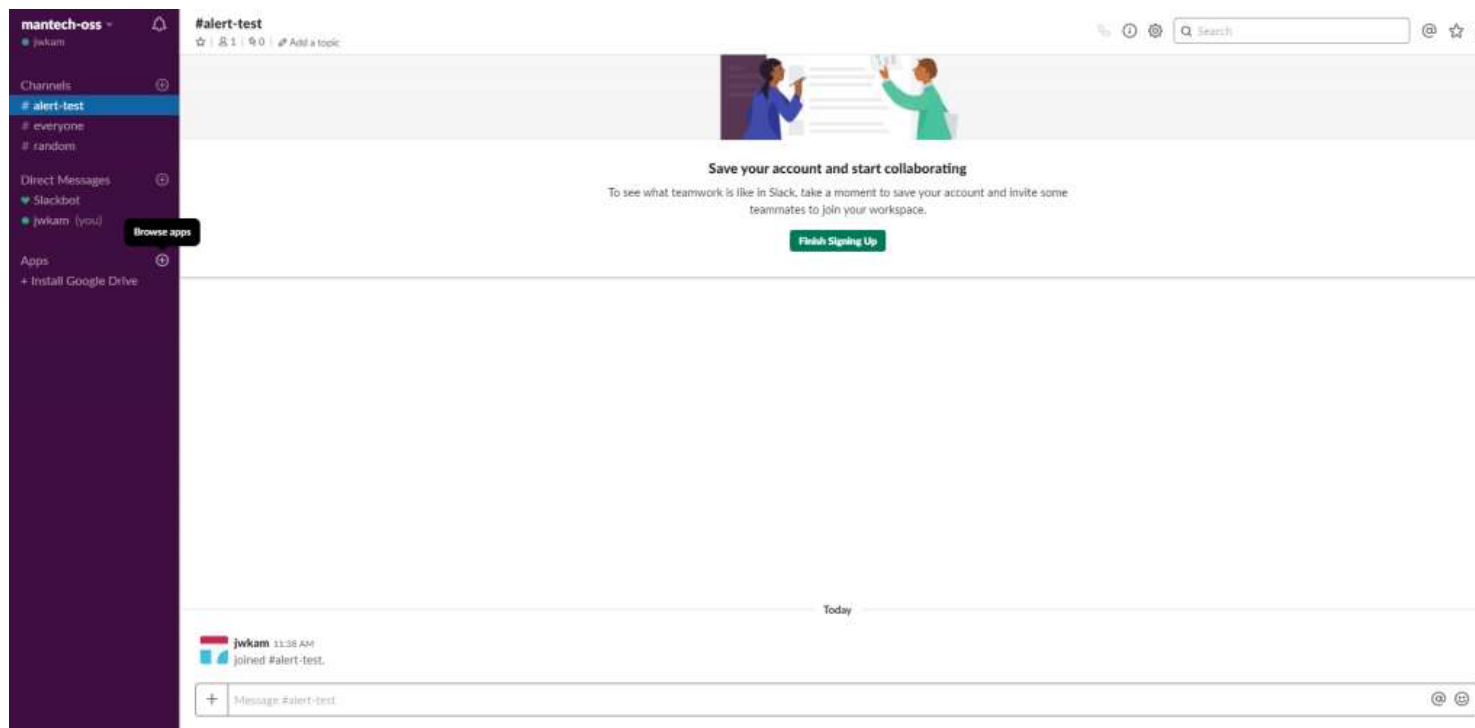
- Slack 생성 (7)



- 채널생성이 완료되면 다음과 같은 화면이 표시 됩니다. "See Your Channel in Slack"을 클릭하여 채널로 합니
다

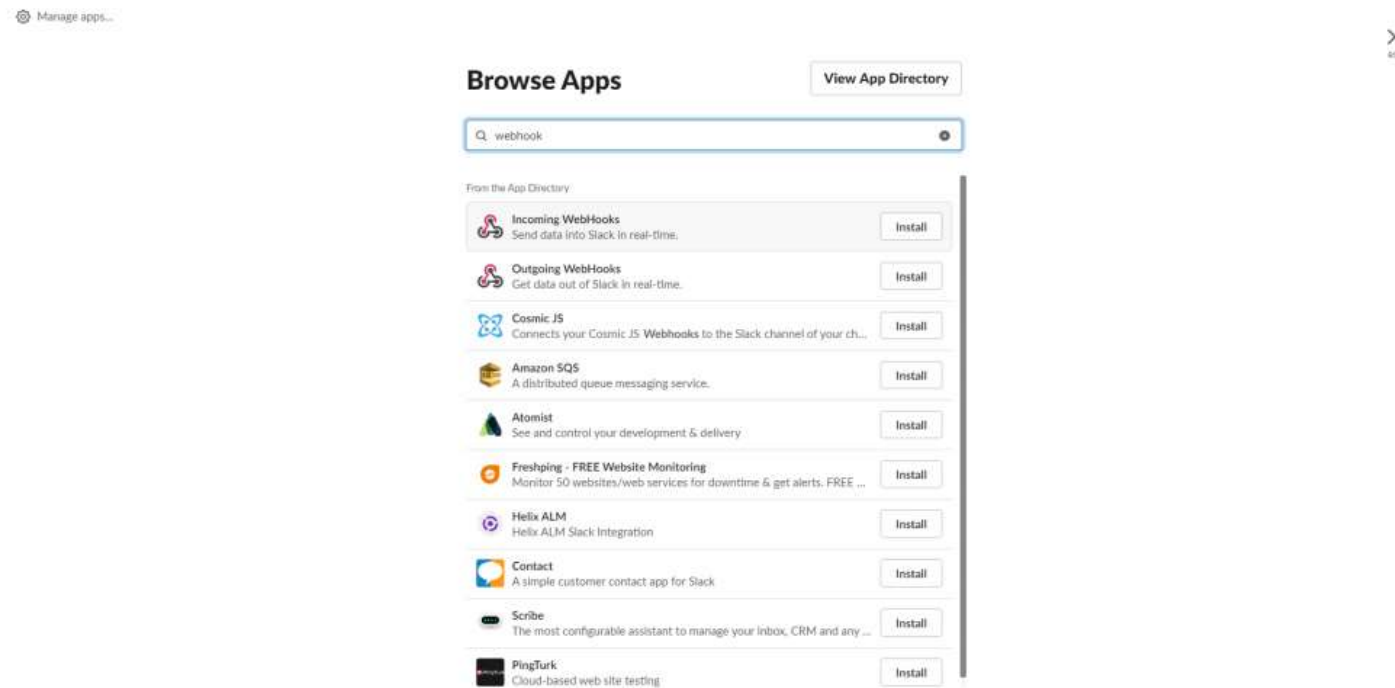
Slack

- Slack 생성 (8)



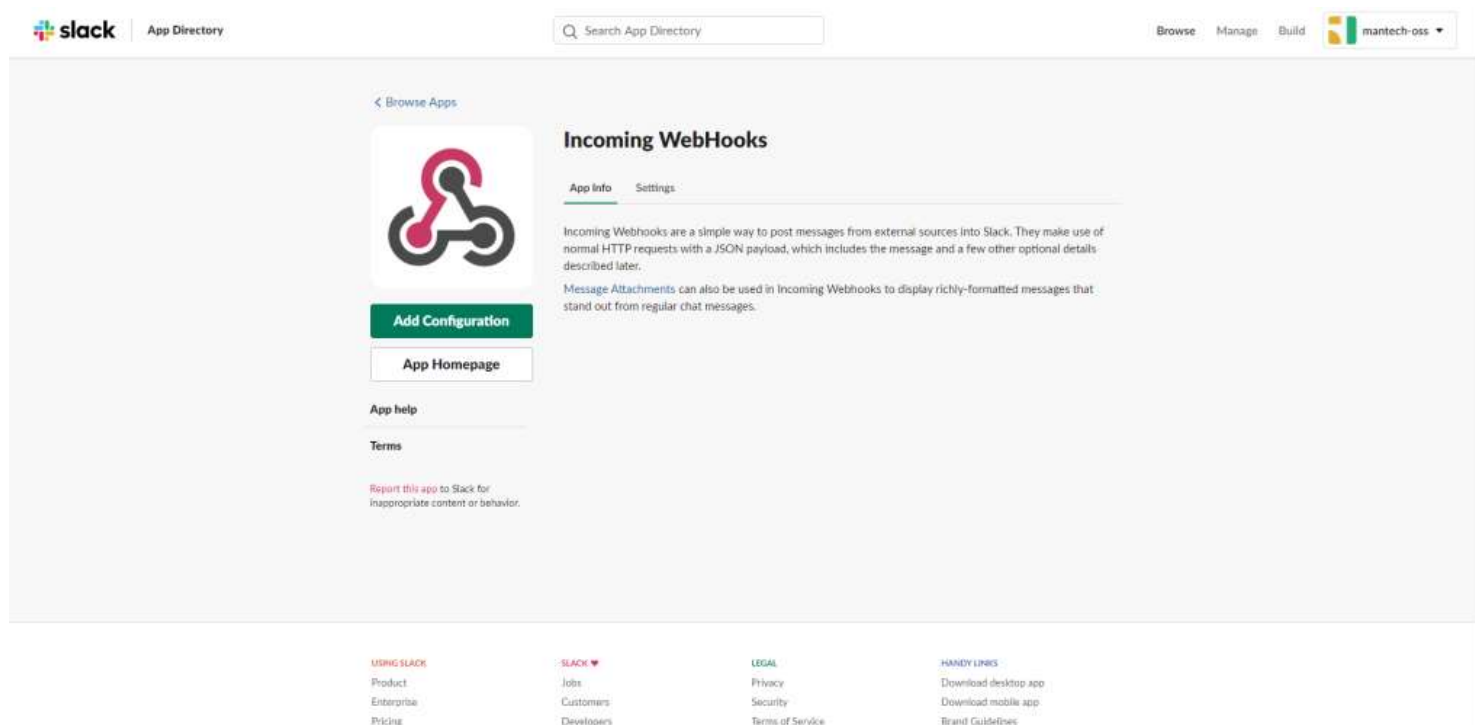
- 채널로 접속 한 화면 입니다. Incoming WebHooks를 설치하면 slack_api_url를 알 수 있습니다.

- Slack 생성 (9)



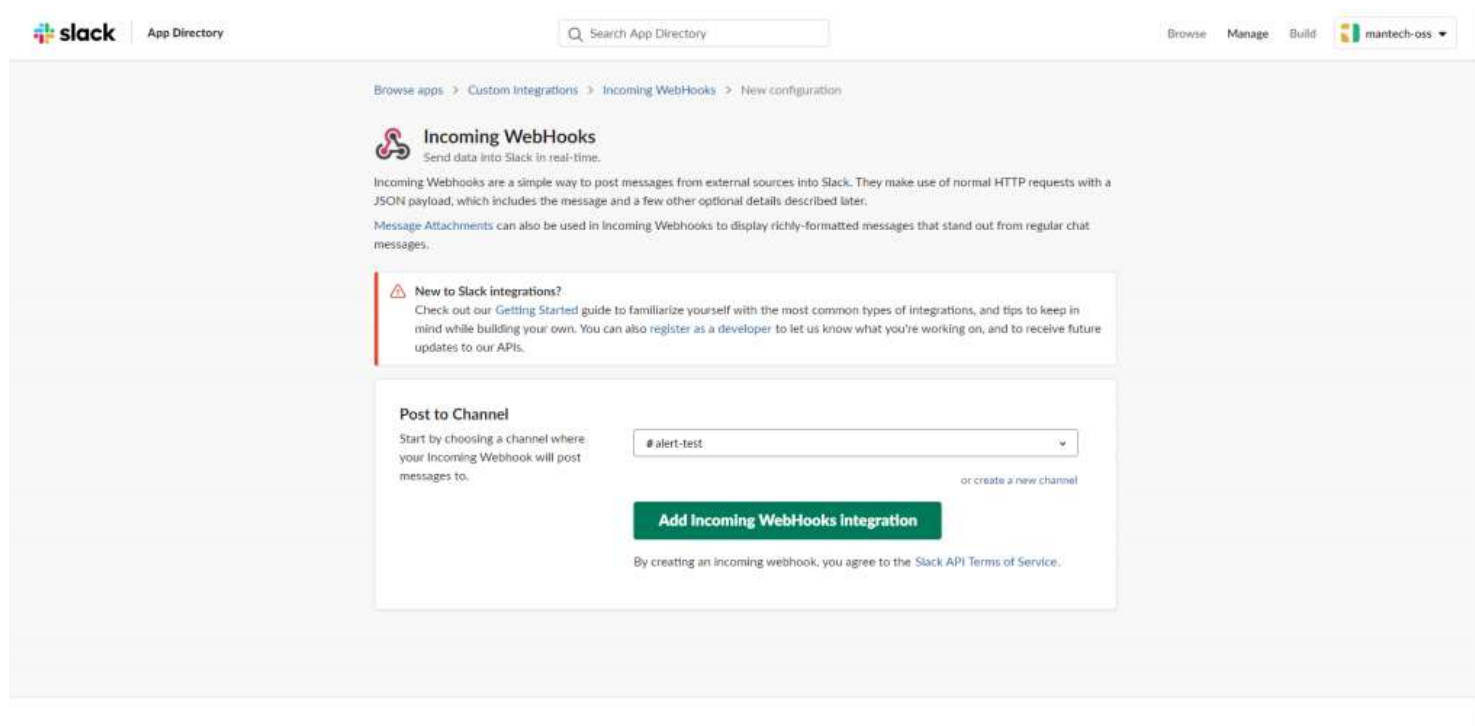
- Browse Apps 화면 상단에 검색창에 WebHook을 검색하여 Incoming WebHooks를 Install 합니다.

- Slack 생성 (10)



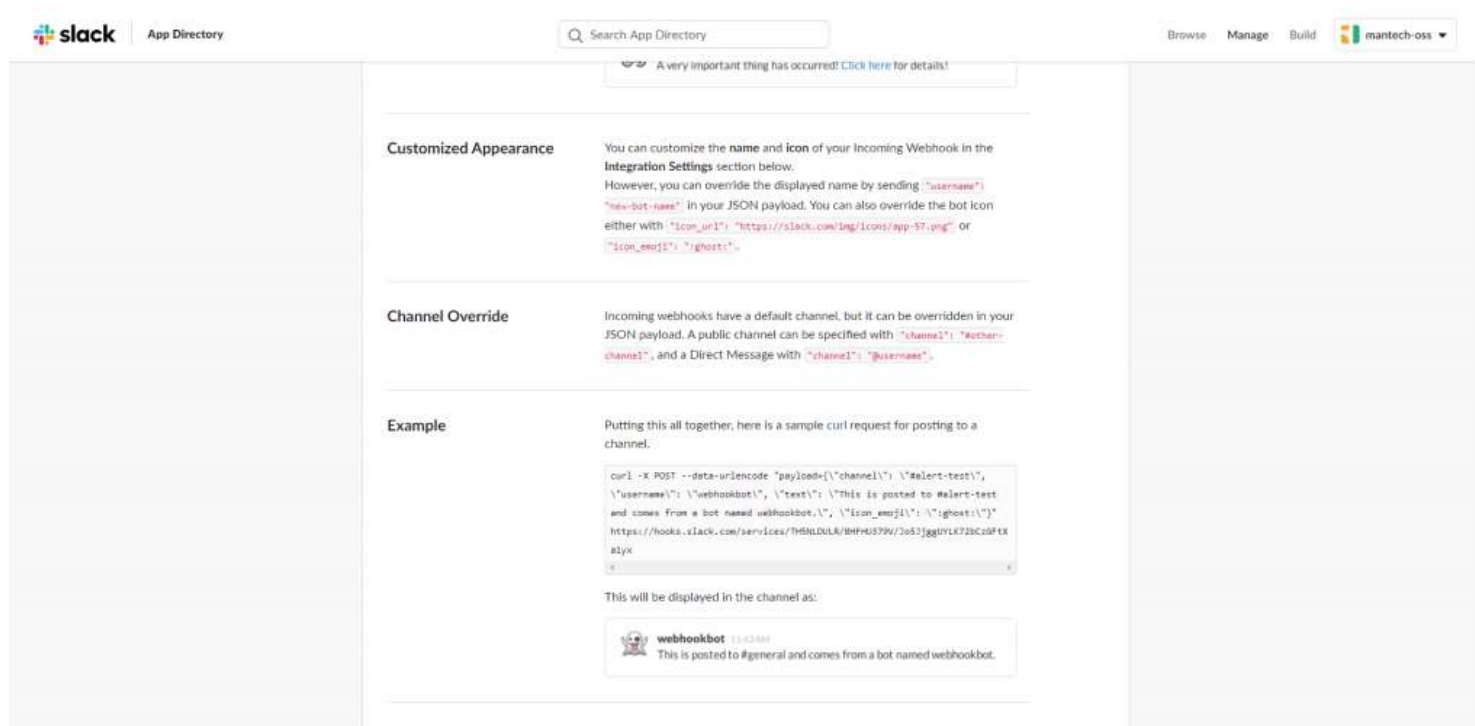
- Add Configuration을 클릭 합니다.

- Slack 생성 (11)



- Post to Channel 에서 채널을 선택 하고 "Add Incoming WebHooks integration"을 클릭 합니다.

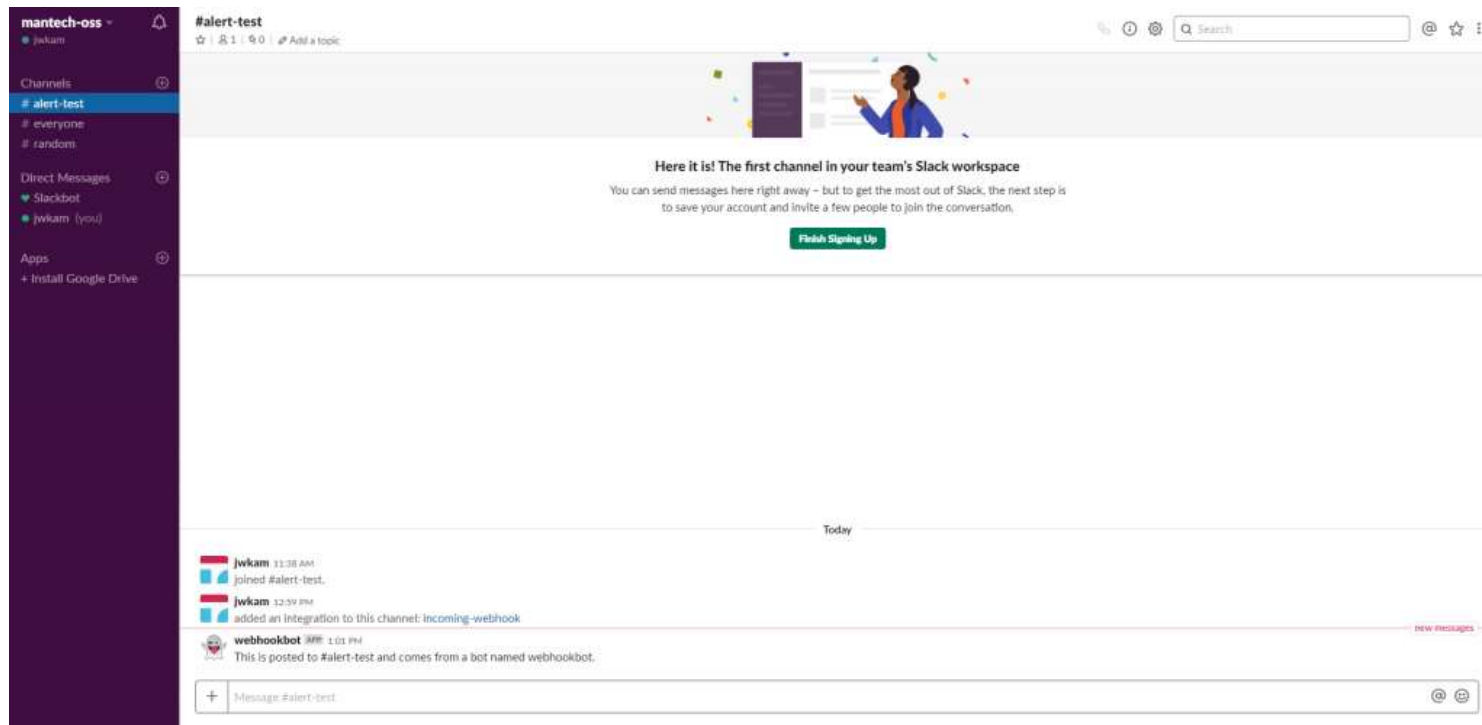
- Slack 생성 (12)



- 생성이 완료 되었으면 “Webhook URL” 주소를 복사하여 메모장에 저장 합니다.
- 아래 Example로 생성된 URL로 메세지가 전송 되는지 테스트 합니다.

Slack

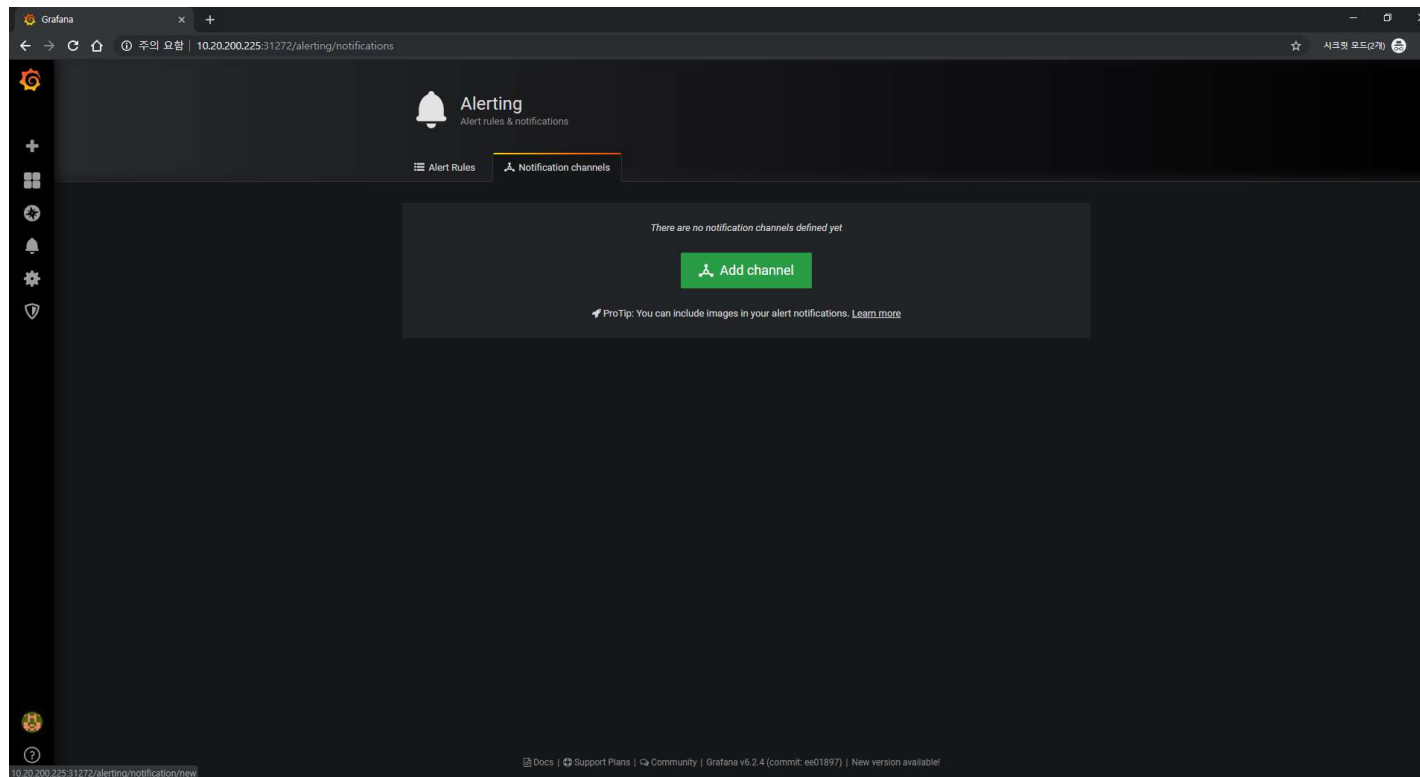
- Slack 생성 (13)



- 정상적으로 구성이 되었으면 아래와 같이 생성된 채널에 메세지가 전송 됩니다.

Alerting

- Alerting - notification channels 추가 (1)



- 생성된 Slack으로 알람을 받기 위해 Grafana에 등록 합니다.
- Grafana 에서 Alerting → notification channels → Add channel 을 클릭 합니다.

Alerting

- Alerting - notification channels 추가 (2)

The screenshot shows the Grafana Alerting interface for configuring a new notification channel. The page title is 'Alerting' with the subtitle 'Alert rules & notifications'. There are two tabs: 'Alert Rules' and 'Notification channels', with the latter being active. In the top right corner, there are two green status bars, each indicating 'Test notification sent' with a checkmark and a close button. The main section is titled 'New Notification Channel' and contains the following fields and controls:

- Name:** A text input field containing 'jwkam-slack'.
- Type:** A dropdown menu set to 'Slack'.
- Default (send on all alerts):** A toggle switch that is currently turned off.
- Include image:** A toggle switch that is currently turned on.
- Disable Resolve Message:** A toggle switch that is currently turned off.
- Send reminders:** A toggle switch that is currently turned off.

Below these settings is a section titled 'Slack settings' with the following fields:

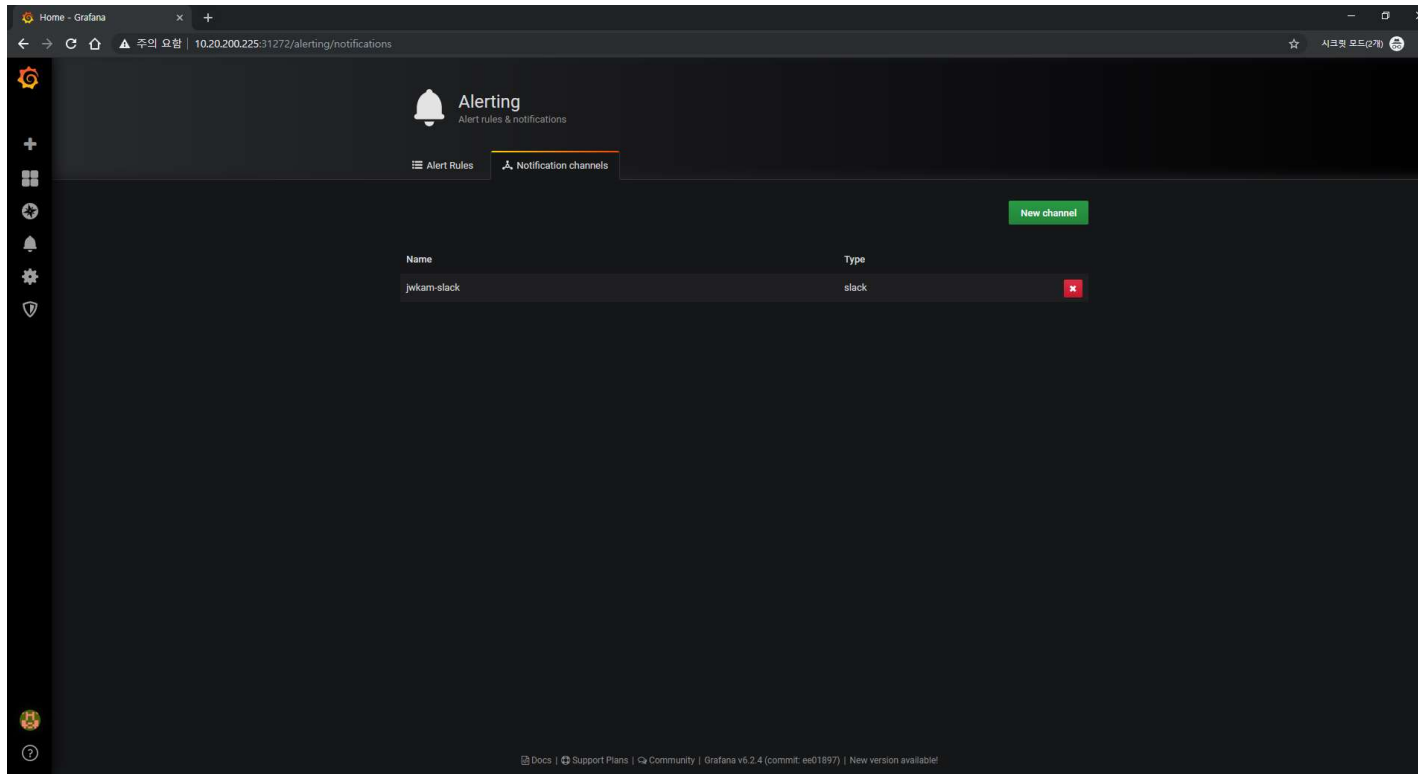
- Url:** A text input field containing 'https://hooks.slack.com/services/TFF33HWDN/...'.
- Recipient:** A text input field with an information icon.
- Username:** A text input field with an information icon.
- Icon emoji:** A text input field with an information icon.
- Icon URL:** A text input field with an information icon.
- Mention:** A text input field with an information icon.
- Token:** A text input field with an information icon.

At the bottom of the form, there are three buttons: 'Save' (green), 'Send Test' (blue), and 'Back' (grey). The footer of the page includes links for 'Docs', 'Support Plans', 'Community', and version information: 'Grafana v6.2.4 (commit: ee01897) | New version available!'.

- **Name:** 식별 가능한 이름 / **Type:** Slack / **Url:** 생성된 Webhook URL
- 입력 후 **Send Test** 버튼을 클릭하여 Slack에서 정상적으로 메시지를 받는지 확인 후 **Save**를 클릭 합니다.

Alerting

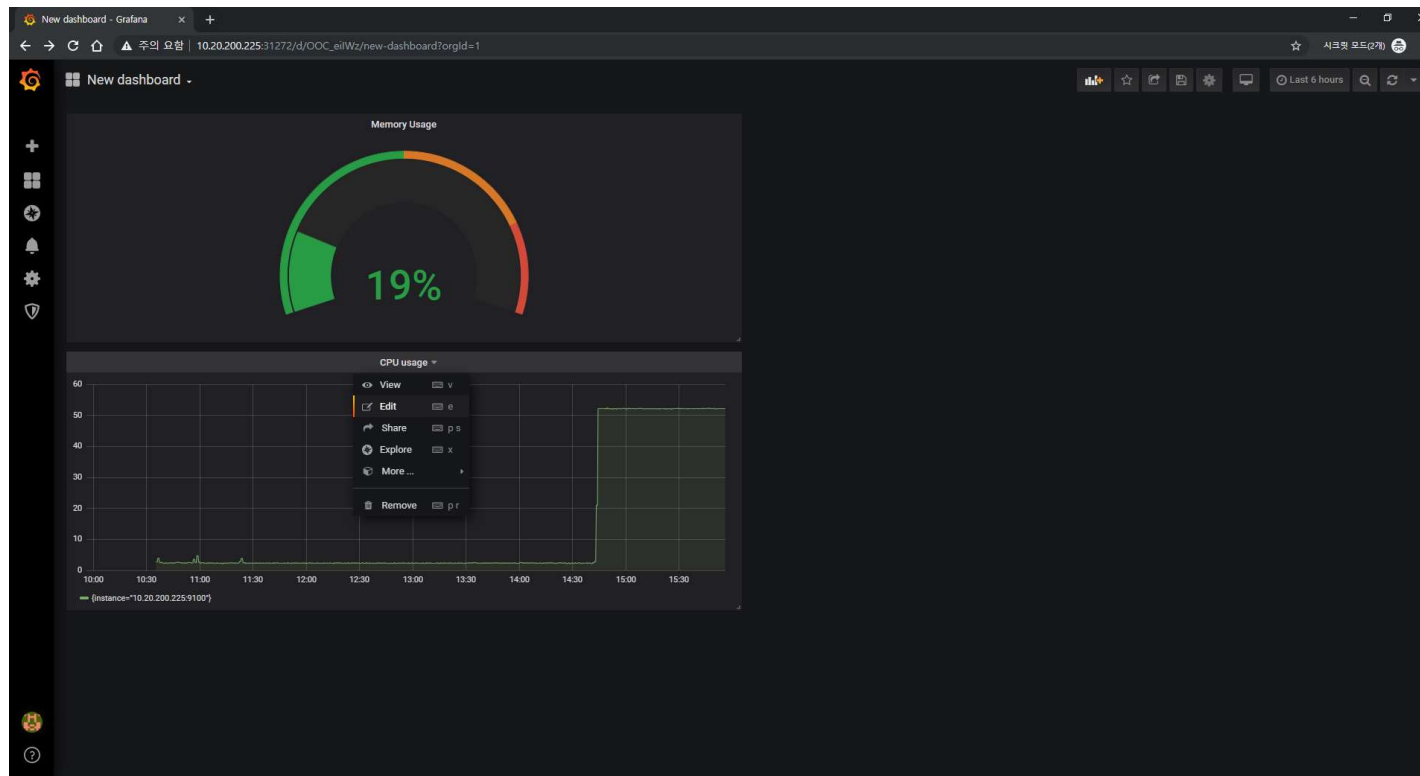
- Alerting - notification channels 추가 (3)



- 저장을 하면 Notification Channels에 추가한 채널이 표시 됩니다.

Alerting

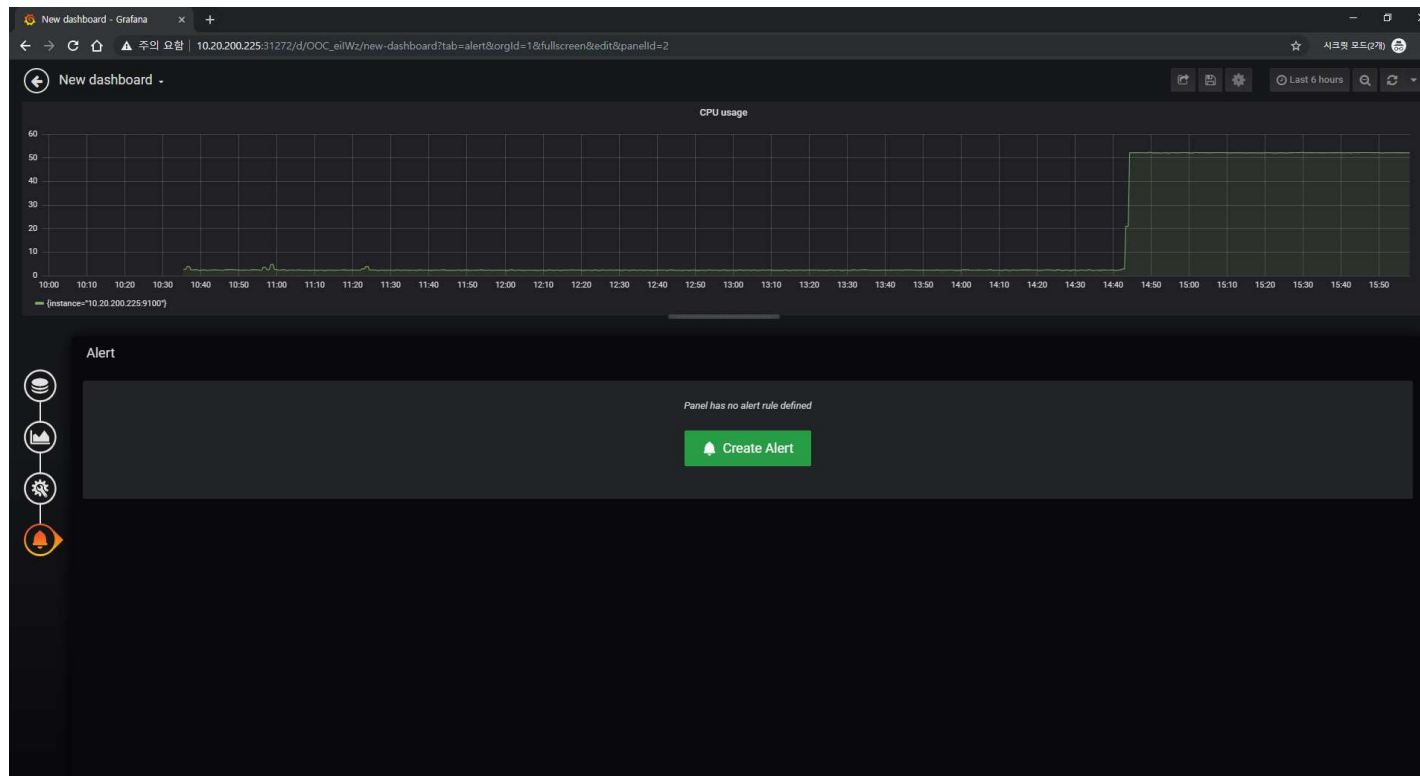
- CPU Usage 알람 설정 (1)



- 대시보드에서 알람 설정을 위해 CPU usage에서 Edit를 선택 합니다.

Alerting

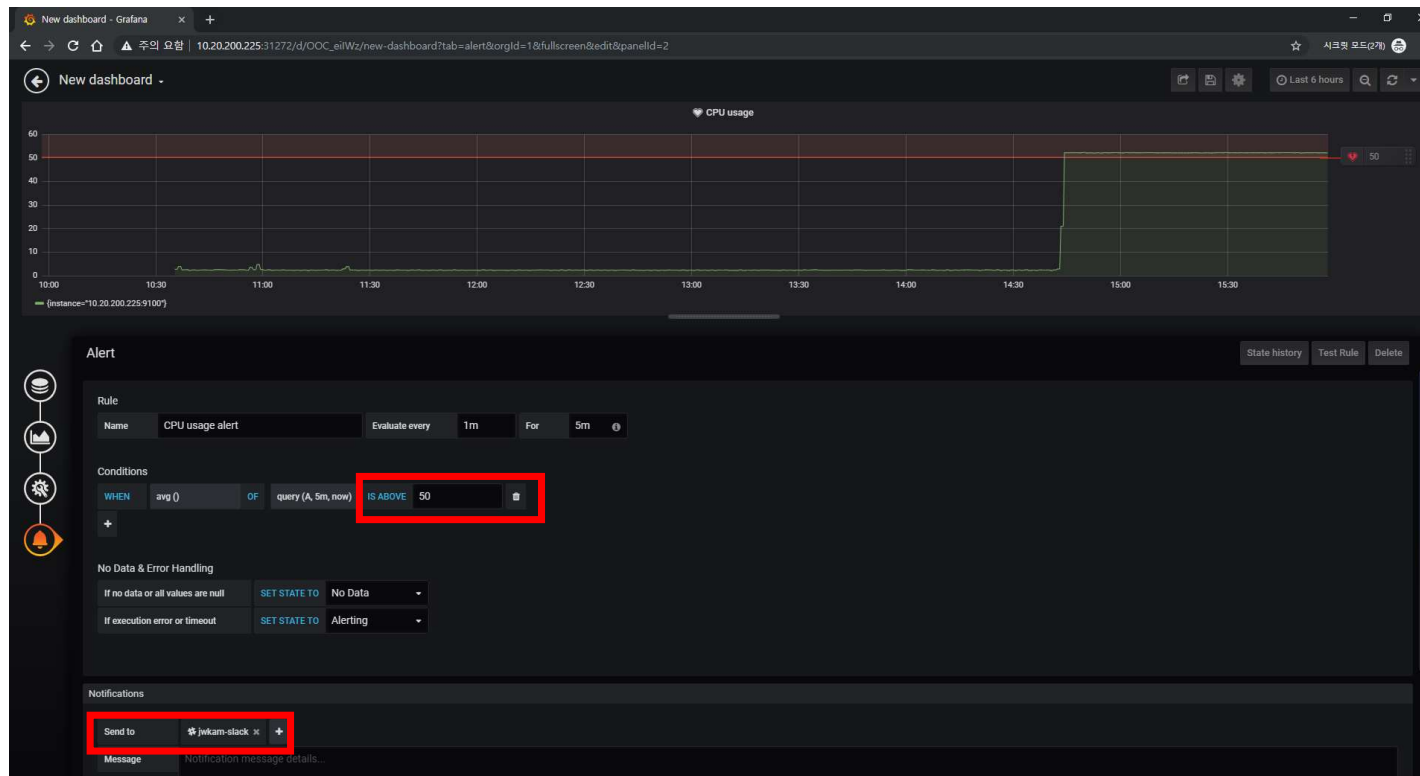
- CPU Usage 알람 설정 (2)



- Alert 설정에서 Create Alert를 선택 합니다.

Alerting

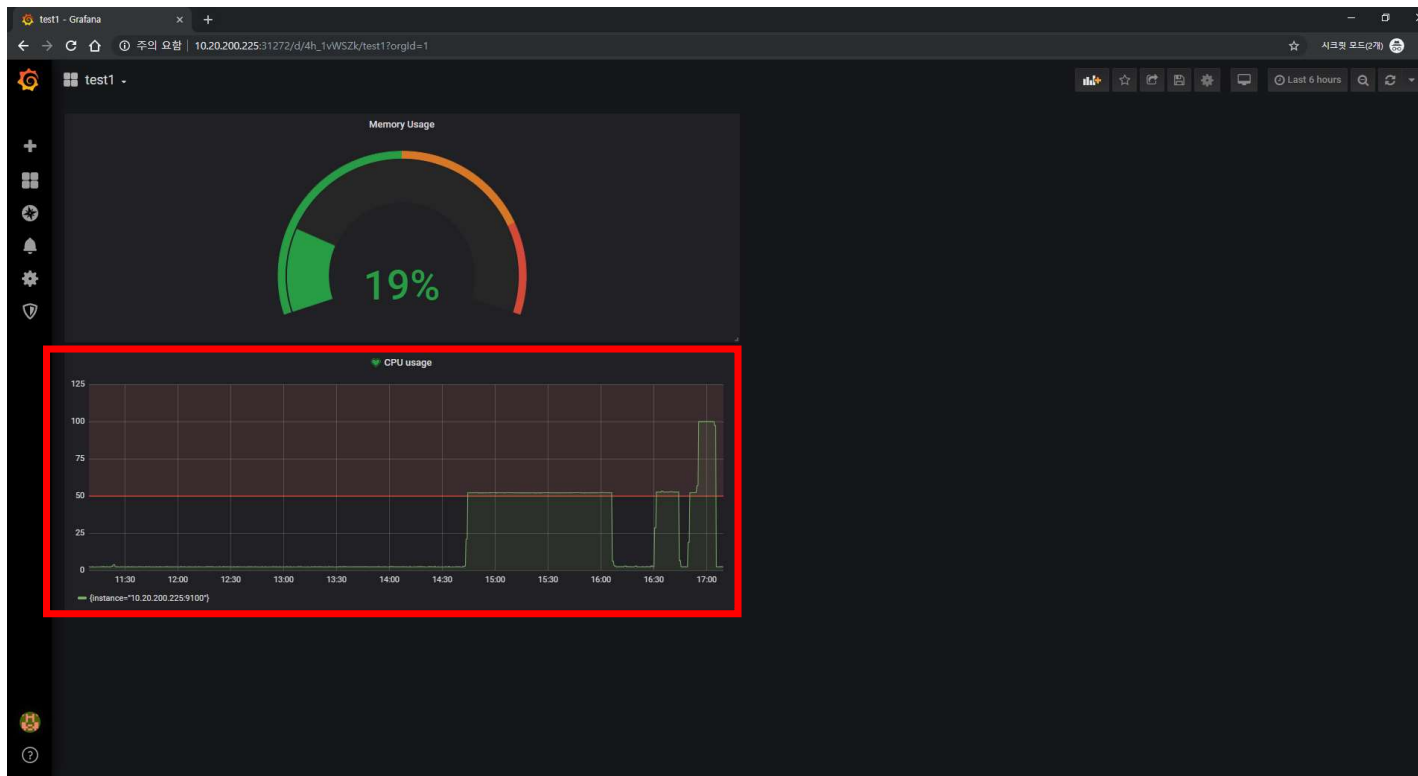
- CPU Usage 알람 설정 (3)



- Conditions 설정에서 IS ABOVE: 50으로 설정 합니다. (테스트를 위해 현재 CPU사용량 이하로 설정)
- Notifications 설정에서 알람을 수신할 채널을 선택 합니다.

Alerting

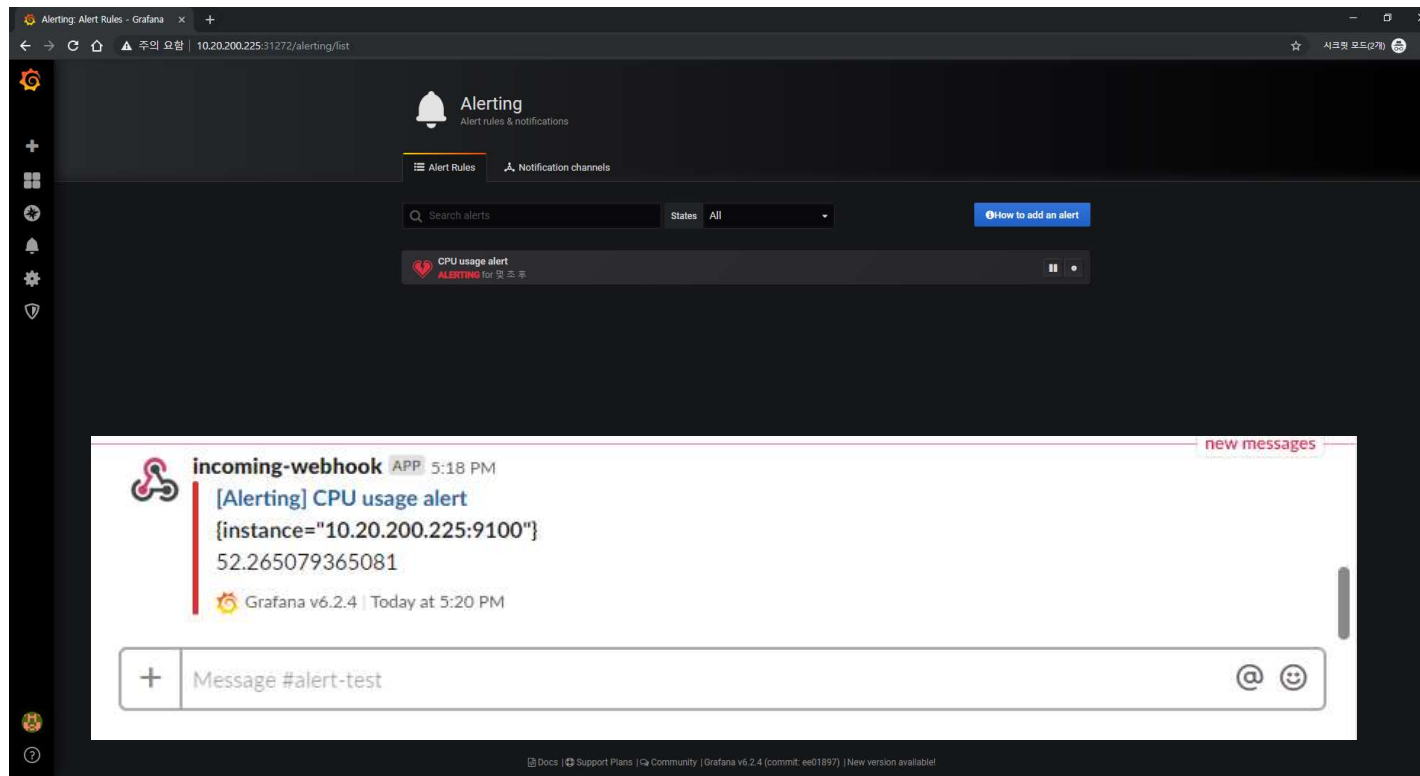
- CPU Usage 알람 설정 (4)



- 알람 설정이 완료 되면 Dashboard에 다음과 같이 변경 됩니다.

Alerting

- CPU Usage 알람 설정 (5)



- Alerting 메뉴에서 Alert상태를 확인이 가능하며 Slack 채널에 받은 알람을 확인 할 수 있습니다.

HPA

(Horizontal Pod Autoscaler)

Prometheus Adapter for Kubernetes Metrics APIs

- Metrics Server가 제공하지 않는 모니터링 데이터를 수집하기 위해 custom metrics를 사용해야 한다.
- Prometheus Adapter는 추가 Kubernetes API를 설치하고 custom metrics API서버를 Kubernetes 클러스터에 등록한다
- Kubernetes resource metrics API 와 custom metrics API가 포함되어 있다.
- Prometheus Adapter는 autoscaling/v2 HPA를 통해 custom metrics API 기반 HPA 사용이 가능하다.
- 리소스 전용 메트릭을 제공하는 metrics-server를 보완하며 대체 가능하다.

HPA

- Prometheus-adapter 설치

```
$ helm install stable/prometheus-adapter --name prometheus-adapter --namespace monitoring --set prometheus.url=http://prometheus-server.monitoring.svc.cluster.local --set prometheus.port=80
```

- Prometheus-adapter가 기본으로 제공하는 custom metrics 확인

```
$ kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1 | jq . |grep "pods"
  "name": "pods/memory_failures",
  "name": "pods/cpu_usage",
  "name": "pods/tasks_state",
  "name": "pods/kube_pod_labels",
  "name": "pods/memory_failcnt",
  "name": "pods/kube_pod_container_resource_requests",
  "name": "pods/cpu_load_average_10s",
  "name": "pods/memory_usage_bytes",
```

HPA

- hpa-example-service 생성

```
$ cat <<EOF > hpa-example-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: hpa-example
spec:
  ports:
    - port: 80
      nodePort: 31002
      targetPort: http-port
      protocol: TCP
  selector:
    app: hpa-example
  type: NodePort
EOF

$ kubectl create -f hpa-example-service.yaml
```

HPA

- hpa-example-deployment 생성

```
$ cat <<EOF > hpa-example-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hpa-example
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: hpa-example
    spec:
      containers:
      - name: hpa-example
        image: gcr.io/google_containers/hpa-example
        imagePullPolicy: IfNotPresent
        ports:
        - name: http-port
          containerPort: 80
      resources:
        requests:
          cpu: 200m
EOF
$ kubectl create -f hpa-example-deployment.yaml
```

HPA

- hpa-example-hpa 생성

```
$ cat <<EOF > hpa-example-hpa.yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-example-autoscaler
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: hpa-example
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Pods
    pods:
      metricName: memory_usage_bytes
      targetAverageValue: 104857600
EOF

$ kubectl create -f hpa-example-hpa.yaml
```


HPA

- 생성된 정보 확인

```
$ kubectl get svc
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
hpa-example         NodePort    10.102.157.108 <none>         31002:31002/TCP  2m40s
kubernetes          ClusterIP   10.96.0.1      <none>         443/TCP          36m

$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
hpa-example-84db48b79c-ctm4        1/1     Running   0          2m38s
hpa-example-84db48b79c-hcksq       1/1     Running   0          2m38s
hpa-example-84db48b79c-w7gsl       1/1     Running   0          2m38s

$ kubectl get hpa
NAME                                REFERENCE          TARGETS          MINPODS  MAXPODS  REPLICAS  AGE
hpa-example-autoscaler  Deployment/hpa-example  17850368/104857600  2         10        3         84s
```

HPA

- 컨테이너에서 부하 주기

```
# 컨테이너 접속
$ kubectl exec -it { POD NAME } - bash

# stress 설치
$ apt-get update
$ apt-get install stress

# 메모리 부하 주기
$ stress --vm 1 --vm-bytes 400M --timeout 2m
```

```
$ kubectl get hpa -watch
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
hpa-example-autoscaler	Deployment/hpa-example	<unknown>/104857600	2	10	0	8s
hpa-example-autoscaler	Deployment/hpa-example	<unknown>/104857600	2	10	3	15s
hpa-example-autoscaler	Deployment/hpa-example	12099584/104857600	2	10	3	30s
hpa-example-autoscaler	Deployment/hpa-example	12101632/104857600	2	10	2	5m33s
hpa-example-autoscaler	Deployment/hpa-example	125192192/104857600	2	10	2	10m
hpa-example-autoscaler	Deployment/hpa-example	125192192/104857600	2	10	3	10m
hpa-example-autoscaler	Deployment/hpa-example	222150656/104857600	2	10	6	11m
hpa-example-autoscaler	Deployment/hpa-example	222150656/104857600	2	10	7	12m
hpa-example-autoscaler	Deployment/hpa-example	16102741/104857600	2	10	7	12m
hpa-example-autoscaler	Deployment/hpa-example	24090624/104857600	2	10	2	17m