Мониторинг

Не забудь включить запись!

План

- Метрики предоставляемые Kubernetes
- Компоненты мониторинга
- Обзор основных метрик
- Мониторинг дополнительных компонентов
- Prometheus

Метрики и мониторинг

Описание всех компонентов мониторинга, и как его использовать описаны в документе

"Kubernetes monitoring architecture"

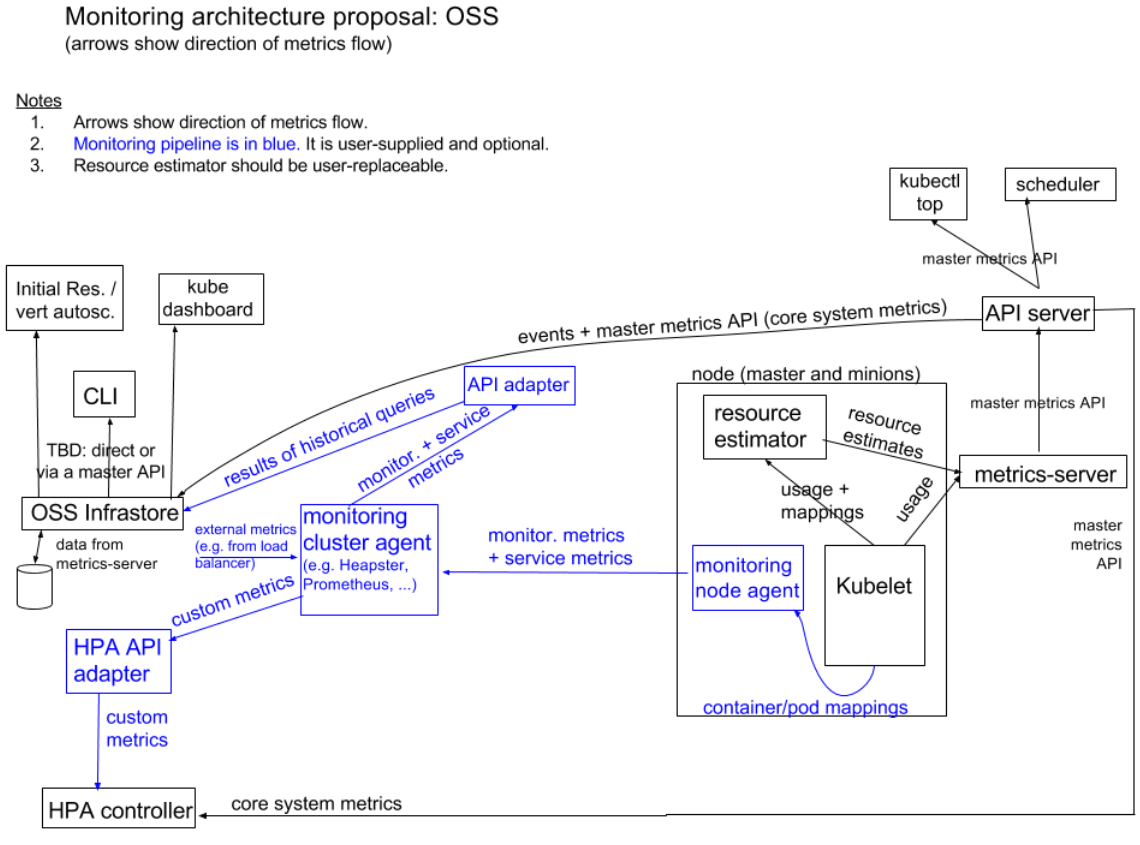
Метрики и мониторинг

- Pipelines
  - Core metrics pipeline
    - Kubelet
    - Resource estimator
    - Metrics-server
    - API server
  - Monitoring pipeline
    - Expose metrics to end-users

Метрики и мониторинг

- Metric-server предоставляет данные для HPA через адаптер
- Использование "Infrastore" в качестве хранилища данных
- Пользовательские системы мониторинга не взаимодействуют напрямую с metric-server
- cAdvisor должен быть источником метрик контейнеров
- Все компоненты k8s должны предоставлять (если должны) метрики в формате пригодном для Prometheus

Метрики и мониторинг



Метрики и мониторинг

Мониторинг нод кластера

- Core metrics
  - CPU
  - RAM
  - Disk
  - Network

Подходы к мониторингу

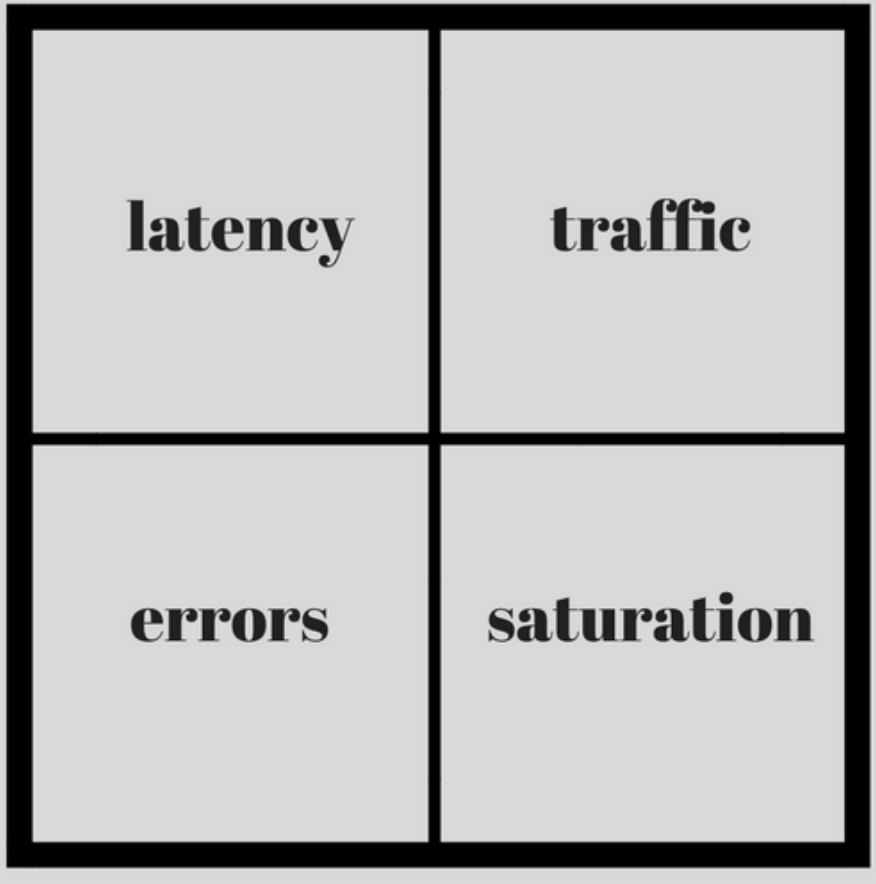
- The Four Golden Signals
- The USE Method
- The RED Method

The Four Golden Signals

Подход был озвучен в "SRE Handbook"

- Latency — Время необходимое на обслуживание запроса
- Traffic — Количество запросов отправляемых на вашу систему
- Errors — Количество ошибок
- Saturation — Насколько полон ваш сервис ^_^

The Four Golden Signals



The USE Method

Подход звучил Брендан Грэг

- Resource - все компоненты физческого\виртуального сервера (CPU, Disk, RAM, etc.)
- Utilization - время которое затрачивает ресурс на выполнение задач
- Saturation - показатель указывающий на количество тасков которые не могут быть выполнены, при этом попадая в очередь
- Errors - количество ошибок

The RED Method

Подход озвучил Том Вилки

```
The USE method is for resources and the RED method is for my services
```

- Rate - количество запросов в секунду
- Errors - количество запросов завершившихся с ошибкой
- Duration - количество время занимаего запросами

Мониторинг CPU

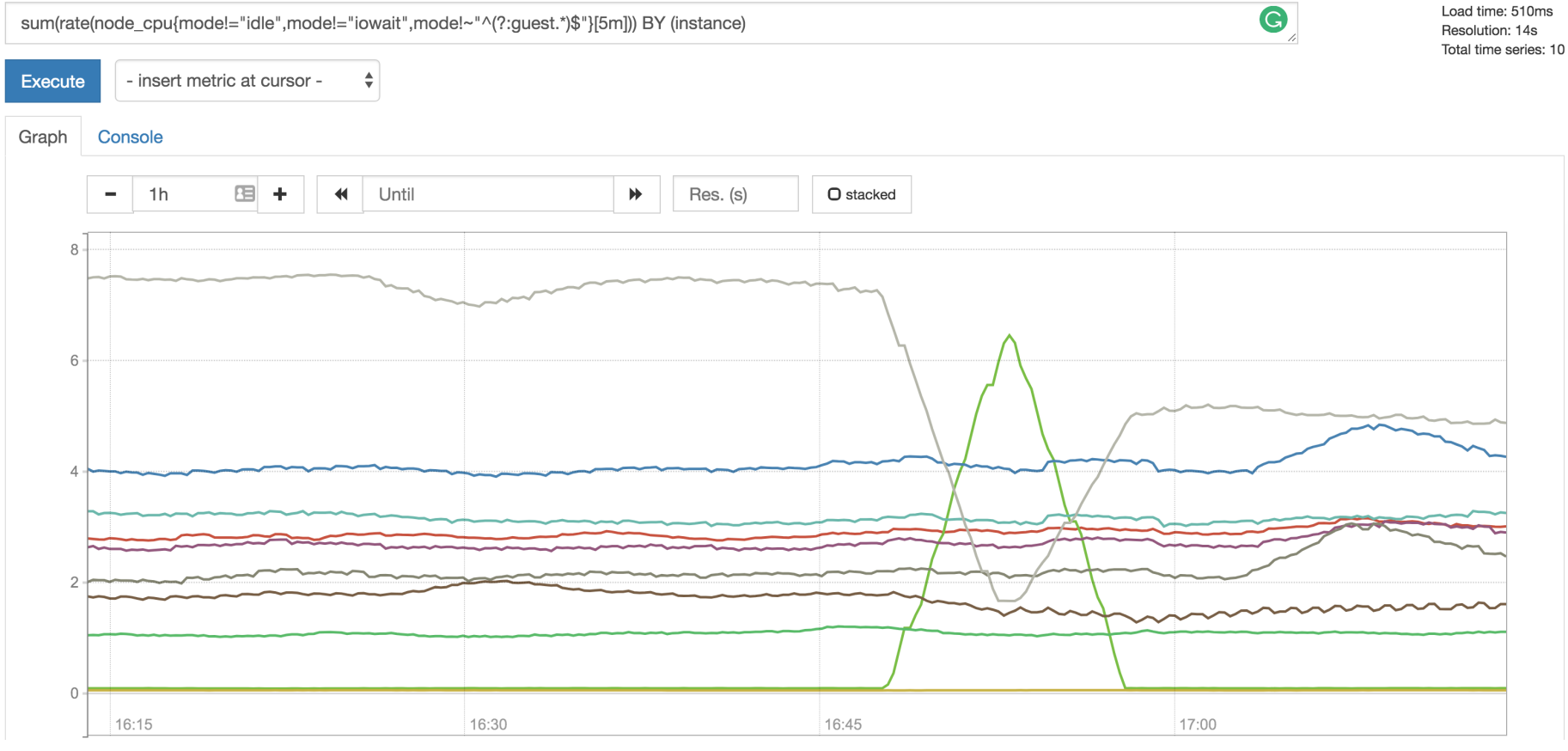
Загрузка CPU для все режимов работы (user, system, nice, idle, iowait, guest, guest_nice, steal, soft_irq, irq)

```
sum(rate(
        node_cpu{mode!=”idle”,
                 mode!=”iowait”,
                 mode!~”^(?:guest.*)$”
                 }[5m])) BY (instance)
```

Суммарное соличество ядер

```
count(node_cpu{mode="system"}) by (node)
```

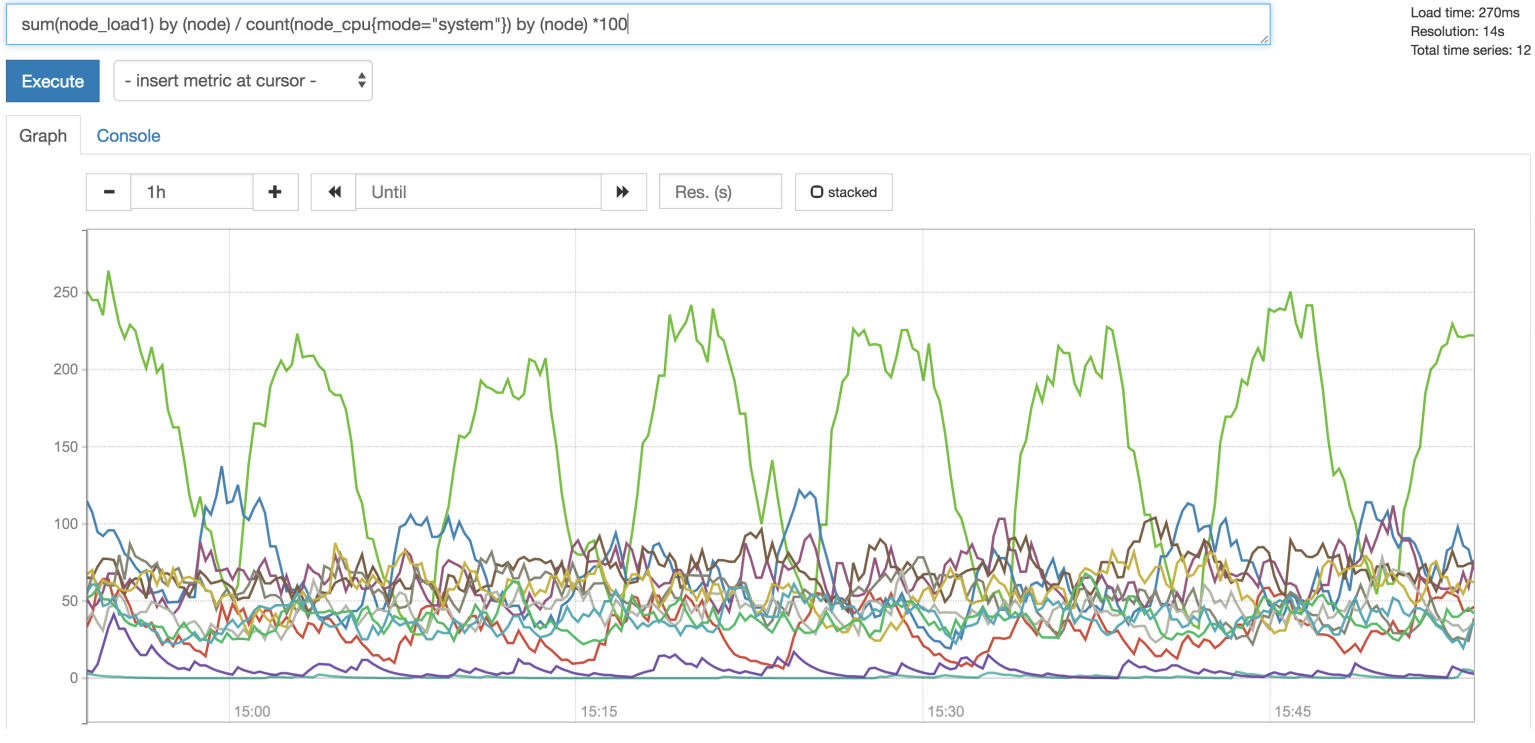
Мониторинг CPU



Мониторинг CPU

```
sum(node_load1) by (node) / count(node_cpu{mode="system"}) by (node) * 100
```

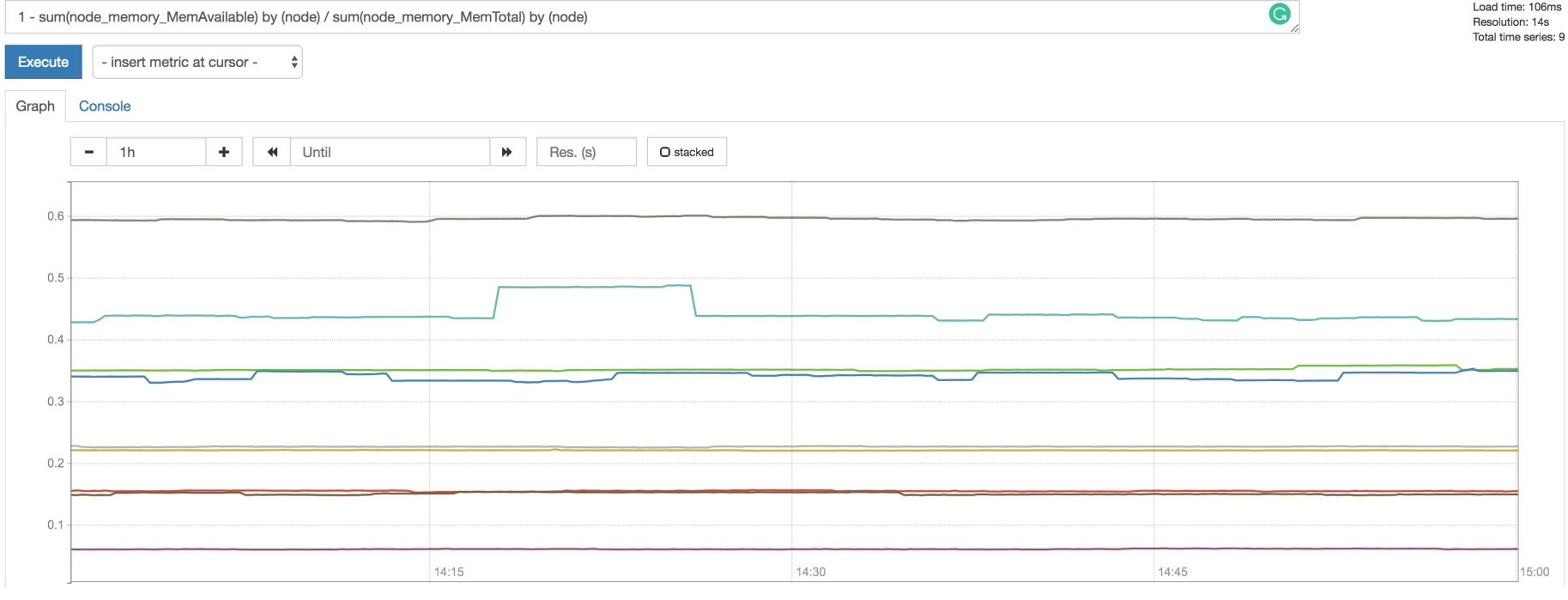
Загрузка нод по количеству CPU



Мониторинг RAM

```
sum(node_memory_MemFree + node_memory_Cached + node_memory_Buffers)
```

```
1 - sum(node_memory_MemAvailable) by (node) / sum(node_memory_MemTotal) by (node)
```



Мониторинг RAM

- node_edac_correctable_errors_total
- node_edac_uncorrectable_errors_total
- node_edac_csrow_correctable_errors_total
- node_edac_csrow_uncorrectable_errors_total

Мониторинг Disk

- Capacity
- Throughput

```
sum(node_filesystem_free{mountpoint="/"}) by (node, mountpoint) /
sum(node_filesystem_size{mountpoint="/"}) by (node, mountpoint)
```

Мониторинг Network

Мониторинг всего трафика

```
sum(rate(node_network_receive_bytes[5m])) by (node) +
sum(rate(node_network_transmit_bytes[5m])) by (node)
```

Мониторинг ошибок

```
sum(rate(node_network_receive_drop[5m])) by (node) +
sum(rate(node_network_transmit_drop[5m])) by (node)
```

Мониторинг (Метрики контейнеров)

- Kubelet предоставляет метрики доступные по /metrics
- Cadvisor => cgroups
- CPU, RAM, DISK, Network

Мониторинг Контейнеры CPU

- container_cpu_user_seconds_total - общее пользовательское время (user ns)
- container_cpu_system_seconds_total - общее системное время (kernel ns)
- container_cpu_usage_seconds_total - сумма

Мониторинг Контейнеры CPU

```
sum(
    rate(container_cpu_usage_seconds_total[5m]))
by (container_name)
```

Мониторинг Контейнеры CPU

- container_cpu_cfs_throttled_seconds_total

```
sum(
    rate(container_cpu_cfs_throttled_seconds_total[5m]))
by (container_name)
```

Мониторинг времени когда наши контейнеры начинают "тротлить"

Мониторинг Контейнеры RAM

- container_memory_working_set_bytes

```
sum(container_memory_working_set_bytes{name!~"POD"})
  by (name)
```

OOMkiller все видит ^_^

Мониторинг Контейнеры DISK

- container_fs_io_time_seconds_total
- container_fs_io_time_weighted_seconds_total

```
sum(rate(container_fs_writes_bytes_total[5m])) by (container_name,device)
sum(rate(container_fs_reads_bytes_total[5m])) by (container_name,device)
```

Мониторинг Контейнеры Network

```
sum(rate(container_network_receive_bytes_total[5m])) by (name)
sum(rate(container_network_transmit_bytes_total[5m])) by (name)
```

- container_network_receive_packets_dropped_total
- container_network_transmit_packets_dropped_total
- container_network_receive_errors_total
- container_network_transmit_errors_total

Мониторинг K8s API Server

- Request Rates and Latencies
- Performance of controller work queues
- Etcd helper cache work queues and cache performance
- General process status (File Descriptors/Memory/CPU Seconds)
- Golang status (GC/Memory/Threads)

Мониторинг K8s API Server

```
- job_name: kubernetes-apiservers
      scrape_interval: 10s
      scrape_timeout: 10s
      metrics_path: /metrics
      scheme: https
      kubernetes_sd_configs:
      - api_server: null
        role: endpoints
        namespaces:
          names: []
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name,
__meta_kubernetes_endpoint_port_name]
        separator: ;
        regex: default;kubernetes;https
        replacement: $1
        action: keep
```

Мониторинг K8s API Server

Запросы: WATCH, PUT, POST, PATCH, LIST, GET, DELETE, and CONNECT

```
sum(rate(apiserver_request_count[5m])) by (resource, subresource, verb)
```

Errors

```
rate(apiserver_request_count{code=~"^(?:5..)$"}[5m]) / rate(apiserver_request_count[5m])
```

Duration

```
histogram_quantile(0.9, sum(rate(apiserver_request_latencies_bucket[5m]))
by (le, resource, subresource, verb) ) / 1e+06
```

Мониторинг K8s API Server

APIServiceRegistrationController

- APIServiceRegistrationController_adds - количество новых событий (rate)
- APIServiceRegistrationController_depth - размер рабочей очереди
- APIServiceRegistrationController_queue_latency - квантили метрики за время проведенное в очереди
- APIServiceRegistrationController_queue_latency_count - количество элементов в очереди (с момента последнего перезапуска)
- APIServiceRegistrationController_queue_latency_sum - все время проведенное в очереди

Мониторинг K8s API Server

- APIServiceRegistrationController_retries - количество попыток перезапуска заданий
- APIServiceRegistrationController_duration - содержит квантили суммарной времени обработки
- APIServiceRegistrationController_duration_count - количество элементов в рабочей очереди
- APIServiceRegistrationController_duration_sum - сумма всех периодов выполнения

Мониторинг K8s API Server

- etcd_helper_cache_entry_count - количество элементов в кэше
- etcd_helper_cache_hit_count - количество запросов в кэш
- etcd_helper_cache_miss_count - количество неудачных запросов к кэшу
- etcd_request_cache_add_latencies_summary - количество времени в мк с на добавление в кэш

Мониторинг K8s API Server

- etcd_request_cache_add_latencies_summary_count - количество элементов в кэше
- etcd_request_cache_add_latencies_summary_sum - все время потраченное на добавление элементов в кэш
- etcd_request_cache_get_latencies_summary - количество времени в мк с затраченное на получение элементов из кэша
- etcd_request_cache_get_latencies_summary_count - суммарное количество запросов GET из кэша
- etcd_request_cache_get_latencies_summary_sum - суммарное количество времени в мк с затраченное на получение элементов из кэша

Мониторинг ETCD

Доступно подробное описание всех метрик

```
- job_name: etcd
  static_configs:
    - targets: ['etcd-a.xxxx.xxxx:4001']
```

- Leader existence and leader change rate
- Proposals committed/applied/pending/failed
- Disk write performance
- Inbound gRPC stats
- Intra-cluster gRPC stats

Мониторинг ETCD

Отслеживание лидера etcd_server_has_leader

```
IF etcd_server_has_leader{job="etcd"} == 0
```

etcd_server_leader_changes_seen_total

```
sum(rate(etcd_server_leader_changes_seen_total[5m]))
```

```
IF increase(etcd_server_leader_changes_seen_total{job="etcd"}[1h])
```

Мониторинг ETCD

- etcd_server_proposals_committed_total
- etcd_server_proposals_applied_total
- etcd_server_proposals_pending
- etcd_server_proposals_failed_total

```
etcd_server_proposals_failed_total{job="etcd"}[1h]) > 5
```

Мониторинг ETCD

- etcd_disk_wal_fsync_duration_seconds_bucket
- etcd_disk_backend_commit_duration_seconds_bucket

```
etcd_disk_wal_fsync_duration_seconds_bucket[5m])) > 0.5
etcd_disk_backend_commit_duration_seconds_bucket[5m])) > 0.25
```

- etcd_grpc_total
- etcd_grpc_requests_failed_total

Мониторинг Kube-state-metrics

- Значения для всех типов объектов
- Все метки и их значения связанные с каждым объектом
- Время создания для всех объектов
- Общая информация специфичная для каждого объекта
- Другие состояния специфичные для определенного объекта

```
count(kube_pod_status_phase{phase="Running"})
count(kube_pod_status_phase{phase="Failed"})
```

Prometheus

Виды федерации:

- Hierarchical federation
- Cross-service federation

Prometheus

```
scrape_configs:
  - job_name: 'federate'
    scrape_interval: 15s

    honor_labels: true
    metrics_path: '/federate'

    params:
      'match[]':
        - '{job="prometheus"}'
        - '{__name__=~"job:.*"}'

    static_configs:
      - targets:
        - 'source-prometheus-1:9090'
        - 'source-prometheus-2:9090'
        - 'source-prometheus-3:9090'
```

Мониторинг и метрики

А какие еще метрики могут быть важны?

Для кого важны?

Мониторинг и метрики

- Платформенная команда
  - Метрики платформы и ее компонентов
  - Возможно общие метрики объектов платформы
- Продуктовая команда
  - Количество ресурсов потребляемых объектом
  - Количество рестартов объекта
  - 5XX
  - Внутренние метрики объекта
- Бизнесс
  - Метрики сделанные исключительно для бизнеса ^_^

Немного про дашборды

Как хранить и распространять дашборды????

Name	Last commit
..	
dashboards	New helmfile
templates	Grafana dashboards chart rename
.helmignore	Grafana dashboards chart rename
Chart.yaml	Grafana dashboards chart rename

Немного про дашборды

```
{{- $files := .Files.Glob "dashboards/*.json" }}
{{- if $files }}
apiVersion: v1
kind: ConfigMapList
items:
{{- range $path, $fileContents := $files }}
{{- $dashboardName := regexReplaceAll "(^.*/)(.*)\\.json$" $path "${2}" }}
- apiVersion: v1
  kind: ConfigMap
  metadata:
    name: {{ $dashboardName | trunc 63 | trimSuffix "-" }}
    labels:
      grafana_dashboard: "1"
  data:
    {{ $dashboardName }}.json: {{ $.Files.Get $path | toJson }}
{{- end }}
{{- end }}
```

Немного про алертинг

..	
rules	New helmfile
templates	Fix prom rules
.helmignore	Infra-service. Init
Chart.yaml	Infra-service. Init

Немного про алертинг

prometheusRules-list.yaml

```
{{- $files := .Files.Glob "rules/*.yaml" }}
{{- if $files }}
apiVersion: v1
kind: List
items:
{{- range $path, $fileContents := $files }}
{{- $prometheusRuleName := regexReplaceAll "(^.*/)(.*)\\.yaml$" $path "${2}" }}
  - apiVersion: monitoring.coreos.com/v1
    kind: PrometheusRule
    metadata:
      name: prometheus-{{ $prometheusRuleName | trunc 63 | trimSuffix "-" }}
      labels:
        app: prometheus
    spec:
      {{- $.Files.Get $path | nindent 6}}
{{- end }}
{{- end }}
```

Немного про алертинг

nodes.rules.yaml

```yaml
groups:
- name: nodes
  rules:
  - alert: NodeCreated
    expr: sum(kube_node_info offset 3m) < sum(kube_node_info)
    for: 1m
    labels:
      severity: warning
    annotations:
      summary: "Cluster autoscaled. New node created"
  - alert: NodeDeleted
    expr: sum(kube_node_info offset 3m) > sum(kube_node_info)
    for: 1m
    labels:
      severity: warning
    annotations:
      summary: "Cluster autoscaled. Node deleted"
```

Немного про алертинг

kubernetes-resources.rules.yaml

```yaml
groups:
  - name: kubernetes-resources
    rules:
    - alert: KubeCPUOvercommit
      annotations:
        message: Cluster has overcommitted CPU resource requests for Pods and cannot
          tolerate node failure.
        runbook_url: https://github.com/kubernetes-monitoring/kubernetes-mixin/tree/master/runbook.md#alert-name-kubecpuovercommit
      expr: |-
        sum(namespace:kube_pod_container_resource_requests_cpu_cores:sum)
          /
        sum(kube_node_status_allocatable_cpu_cores)
          >
        (count(kube_node_status_allocatable_cpu_cores)-1) / count(kube_node_status_allocatable_cpu_cores)
      for: 5m
      labels:
        severity: warning
    - alert: KubeMemOvercommit
      annotations:
        message: Cluster has overcommitted memory resource requests for Pods and cannot
          tolerate node failure.
        runbook_url: https://github.com/kubernetes-monitoring/kubernetes-mixin/tree/master/runbook.md#alert-name-kubememovercommit
      expr: |-
        sum(namespace:kube_pod_container_resource_requests_memory_bytes:sum)
          /
        sum(kube_node_status_allocatable_memory_bytes)
          >
        (count(kube_node_status_allocatable_memory_bytes)-1)
          /
        count(kube_node_status_allocatable_memory_bytes)
      for: 5m
      labels:
        severity: warning
```

Немного про алертинг

prometheus-operator.yaml.gotmpl

```
alertmanagerSpec:
  replicas: 3
resources:
  limits:
    cpu: 20m
    memory: 64Mi
  requests:
    cpu: 10m
    memory: 32Mi
config:
  global:
    resolve_timeout: 5m
  route:
    group_by: ['alertname', 'job']
    group_wait: 30s
    group_interval: 1m
    repeat_interval: 12h
    receiver: 'telegram'
    routes:
    - match:
        severity: warning
      receiver: 'telegram'
    - match:
        severity: high
      receiver: 'telegram'
    - match:
        severity: critical
      receiver: 'telegram'
    - match:
        alertname: Watchdog
      receiver: 'null'
  receivers:
  - name: 'telegram'
    webhook_configs:
    - send_resolved: True
      url: http://prometheus-telegram-bot:9087/alert/{{ requiredEnv "TELEGRAM_ROOM_ID" }}
  - name: 'null'
```

Prometheus хранение метрик

Отдельные хранилища для исторических данных:

- Victoria Metrics
- Cotrex
- Thanos
- Clickhouse ^_^

Prometheus Victoria metrics

Prometheus Victoria metrics

- Поддержка Prometheus querying API (доп расширения)
- Высокий уровень сжатия
- Оптимизация под "медленные" хранилища
- Защита от kill -9 ^_^
- Кластерная версия

Prometheus Cortex

Prometheus Cortex

- Горизонтальное масштабирование
- Высокая доступность
- Multi-tenant
- Хранилище долгосочных данных

Prometheus Thanos

Prometheus Thanos

- Выполнение запросов ко всем подключенным prometheus из одной точки
- Проверка и слияние метрик полученных от инсталляций prometheus HA
- Ускорее запросов к "старым" данным
- Кросс-кластерная федерация
- Отказоустойчивая маршрутизация запросов
- gRPC в качестве Store API для доступа ко всем собранным метрикам

Prometheus Thanos