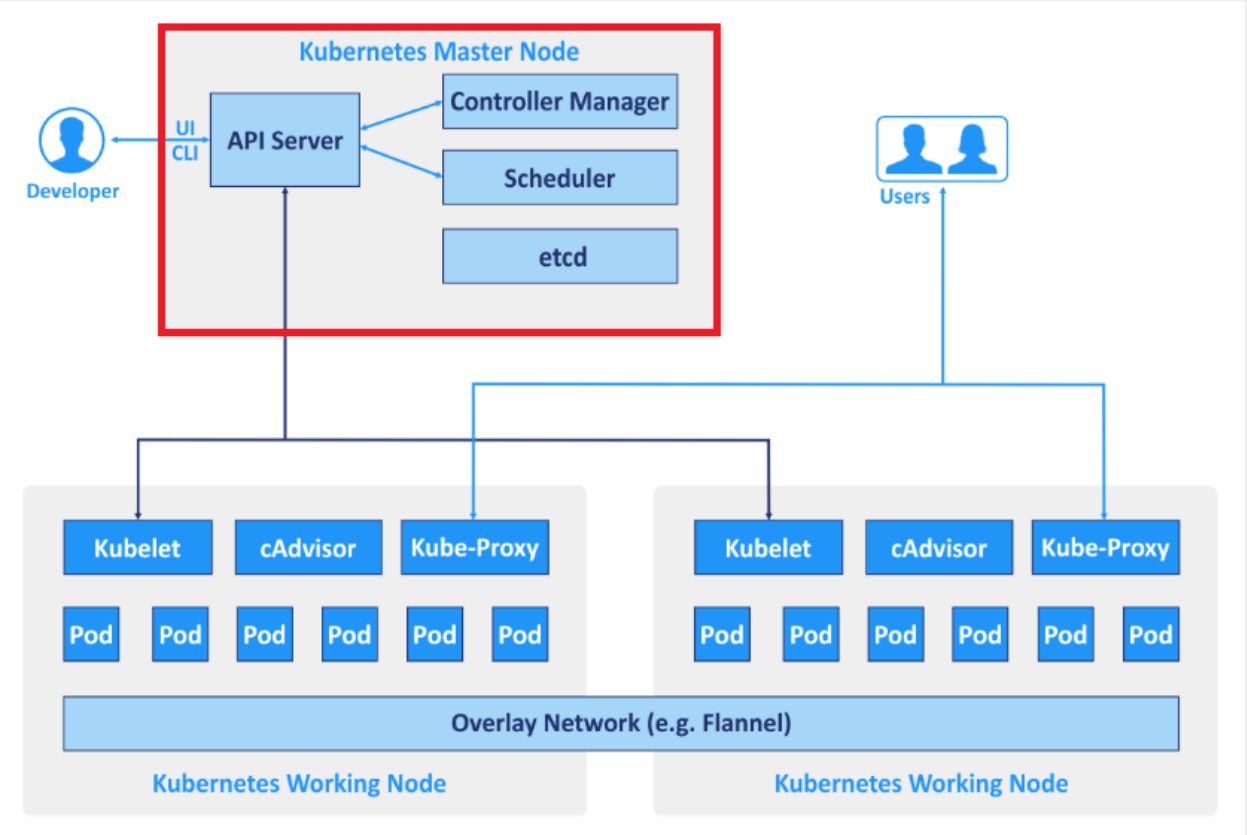
مانند بیشتر پلتفرم های توزیع یافته، Kubernetes cluster هم حداقل از یک master و یک یا چند نود node ساخته شده است.

در بسیاری از کتاب ها و سایت های از node در kubernetes به عنوان یک worker یاد میشود که در گذشته به آن minion گفته میشد.همانطور که از اسامی پیداست master در حقیقت نقش مدیریتی و رهبری را بازی میکند و وظیفه ی اصلی آن گرفتن تصمیمات در کلاستر میباشد و در مواقعی که پیشامدی رخ میدهد بتواند بهترین رویکرد ها را ارایه دهد از master به نام Control plane node هم یاد میشود. Maser و Worker هر کدام از Component های مختلف تشکیل شده است که در این داکیومنت سعی داریم به تعریف و بررسی کوتاه master components بپردازیم.

Master در kubernetes از چهار component تشکیل شده است:

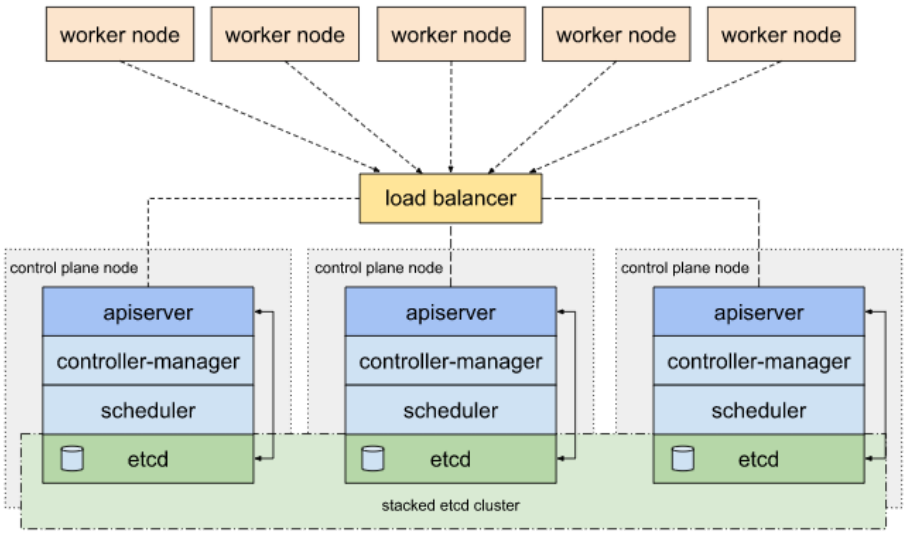
- etcd
- API Server
- Scheduler
- Controller



این چهار component به کمک یک دیگر میتوانند قدرت تصمیم گیری و بررسی شرایط را برای master بوجود آورند.

## Etcd

یکی از نیاز های مهم در محیط های کلاستری که چند ماشین/نود در حال کار کردن با هم هستند تا بتوانند یک یا چند سرویس مختلف را ارائه بدهند این است که توانایی دسترسی به یک سری اطلاعات مهم در مورد یک دیگر و اجزاء تشکیل دهنده ی کلاستر را داشته باشند.این اطلاعات باید در دیتابیسی ذخیره شود و طبق قوانین خاص در دسترس node ها قرار گیرد برای این نیاز etcd به میدان بازی می آید.




etcd در حقیقت یک محل ذخیره سازی توضیع شده بر اساس کیلید میباشد(Distributed key values store) که در بسیاری از محیط های cluster شده مانند kubernetes استفاده میشود.در زمانی که بخواهیم Hight Availibiliy هم در کلاستر خود داشته باشیم باید etcd cluster راه اندازی کنیم.همانطور که در شکل بالا میبینیم 3 تا master یا همان Control plane node داریم که در پایین ترین لایه خود همگی از یک etcd که به صورت clusterدر آمده است استفاده میکنند.

برای درک بهتر etcd بهتر است نگاهی بیندازیم به اطلاعاتی که در آن ذخیره میشود:

- اطلاعات پیکربندی کلاستر
- حالت فعلی کلاستر(actual state) و حالت دلخواه(desired state)
- اطلاعات تمامی اشیاء موجود در کلاستر
- اطلاعات در مورد node ها و وضعیت آنها

در زمانی که ما از دستور kubectl get <something> استفاده میکنیم در حقیقت در حال خواندن اطلاعات موجود در etcd هستیم و به زبان دیگر از etcd میخواهیم اطلاعاتی در مورد موجودیتی را به ما بدهد و در زمانی که با دستور kubectl create <something> موجودیتی را میسازیم در حقیقت اطلاعات جدیدی را در etcd ذخیره میکنیم.اطلاعات موجود در etcd بنا به اتفاقات مختلف در حال تغییر هستند به عنوان مثال اگر node A به هر دلیلی از سرویس دهی خارج شود اطلاعات سلامتی این نود در etcd تغییر پیدا میکند.اما سوال اینجاست که etcd از کجا میفهمد که موجودیتی در کلاستر تغییر پیدا کرده است؟

هر تغییر در kubernetes به عنوان یک رویداد(event) به حساب می آید و مفهومی به اسم watch feature مدام در حال مانیتورینگ کلید اشیاء میباشد و زمانی که قرار است به هر دلیلی رویدادی رخ دهد و اطلاعات هر کلیدی تغییر پیدا کند ، watch متوجه آن خواهد شد.باید در نظر داشت watch feature یکی از مهم ترین ویژگی های موجود در etcd است به این دلیل که API Server هم بشدت برای اعمال تغییرات و اتخاذ تصمیمات مناسب متکی به watch feature میباشد.

از آنجایی که تمام اطلاعات مهم در etcd ذخیره میشود بک آپ گیری از etcdباید یکی از موارد مورد توجه شما باشد.

## API Server

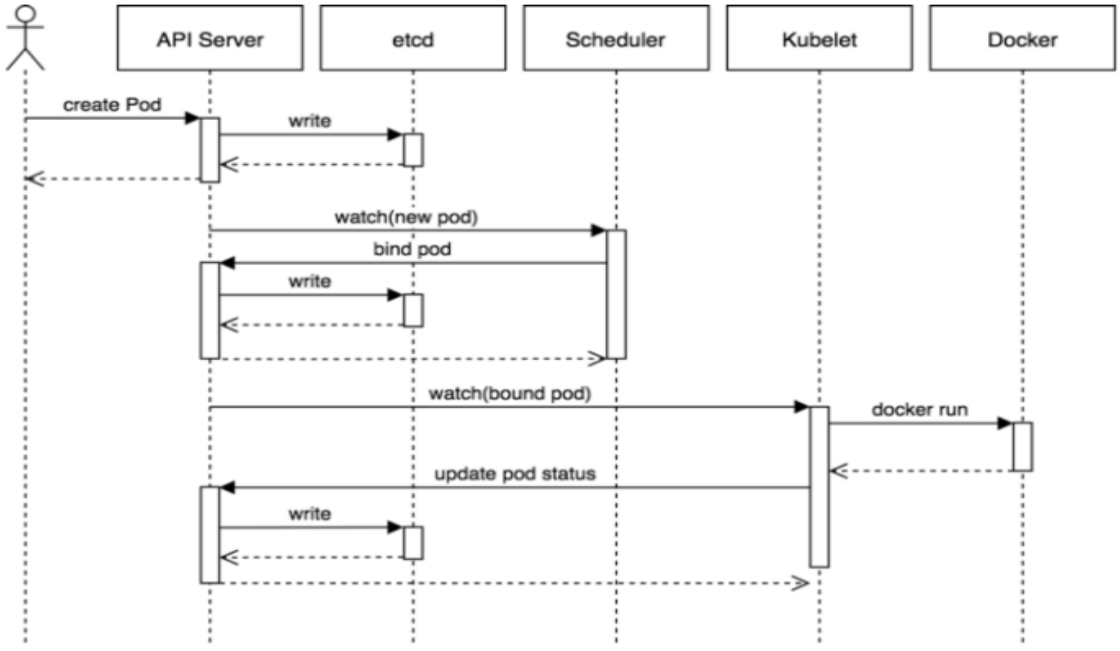
از API Serverمیتوان به عنوان درگاه ورودی به کلاستر kubernetes یاد کرد چرا که تمامی دستورات REST در این کامپوننت کامپایل میشود.این کامپوننت مسولیت مهم مدیریت مرکزی kubernetes را بر عهده دارد. API Serverتنها کامپوننتی هست که

اجازه ی دسترسی مستقیم به etcd را دارد و بقیه کامپوننت ها از طریق API Server میتوانند به اطلاعات etcd دسترسی پیداکنند.

زمانی که ما به عنوان یک admin/client با دستور kubectl درخواست خود را مطرح میکنیم در حقیقت در حال برقرای ارتباط با API Server میباشیم.درخواست ما در سمت API Server به عنوان یک REST Operation درنظر گرفته میشود و این درخواست میتواند گرفتن،تغییر دادن،ساختن و یا پاک کردن یک Object در Kubernetes باشد.در صورتی که تغییرات مدنظر یک موجودیت موفقیت آمیز باشد اطلاعات آن موجودیت در etcd تغییر پیدا میکند.

همانند etcd در apiserver هم ما مفهومی به اسم watchرو به رو هستیم که وظیفه اصلی آن برقراری ارتباط بین API Server با بقیه کامپوننت ها در زمان notify شدن است.

بگزارید نقش مهم و اساسی API Serverرا در زمان ساخت یک pod با مثال و شکل زیر مرور کنیم:




وقتی کاربری دستور ساخت یک pod را با kubectl میزند این درخواست از طریق API Server به عنوان یک REST operation در نظر گرفته میشود و اولین کار این هست که این درخواست و اطلاعات مهم موجودیت هایی که در این درخواست درگیر هستند به وسیله ی API Server در etcd ذخیره شود.در حین این پروسه ها watch (در etcd به آن پرداختیم) مدام در

حال مانیتورینگ موجودیت ها میباشد و زمانی که متوجه شود قرار بر تغییر موجودیتی هست به اصطلاح notify میشود.متوجه شدن watch یعنی وظیفه ی اصلی Scheduler شروع شده است بر این اساس که حال Scheduler باید طبق قوانینی نودی را برای pod تازه به دنیا آمده ی ما در نظر میگیرد و آن را به API Server در میان بگزارد.

بعد از مطلع شدن API Server از نودی که قرار است برای pod جدید ما در نظر گرفته شود اطلاعات جدید در etcd نوشته میشود و بعد از آن Kubelet بر روی آن نود فراخوانی میشود.اما kubeletچیست؟ بر روی هر نود یا همان minion یک agent به اسم kubelet در حال فعالیت هست و اصلی ترین وظیفه ی آن اطمینان از سلامت container های آن نود است و یا گرفتن فرمان از سوی API Server و در میان گذاشتن آن با daemon مربوط به کانتینر(که در بیشتر اوقات docker میباشد)آن نود.پس API Server به kubelet نوده انتخاب شده دستور ساخت کانتینر مد نظر را میدهد و kubelet بعد از دریافت این دستورات آن را به docker daemon در آن نود در میان میگزارد و container ساخته میشود. kubelet خبر خوب ساخته شدن container را به API server میدهد و API server هم گزارش تغییرات جدید را در etcd ثبت میکند.

در توضیحات و عکس بالا etcd و watch نقش بسیار مهمی را بازی میکنند.تمام اطلاعات تغییر و تحولات در etcd به دست api server ثبت میشود تا همگی از آن خبر داشته باشند و watch هم نقش پل ارتباطی بین Api Server با باقی کامپوننت ها را بازی میکند.

## Scheduler

در kubernetes واژه ی Scheduling به معنای حصول اطمینان از منطبق شدن Pod با نود انتخاب شده است تا kubelet بتواند pod در خواستی را راه اندازی کند.زمانی که گفته میشود Scheduling به مشکل خورده است به این معناست که pod درخواستی همچنان بدون نود است.مسولیت تخصیص نود به pod تازه متولد شده یا pod ای که به هر دلیلی نودی را ندارد بر عهده ی Scheduler هست. Kube-scheduler برنامه ریز پیش فرض در kubernetes میباش و نکته ی جالب توجه این است که ما میتوانیم scheduler خود را بسیازیم و به جای kuber-scheduler از آن استفاده کنیم.

یک pod از مجموعه ای از container ها ساخته شده است و container دارای نیاز های متفاوتی هست که میتوانیم آن ها را معلوم کنیم به عنوان مثال N میزان RAM لازم است تا فلان containerبه درستی اجرا شود پس میتوانیم نتیجه بگیریم Pod نیاز های خود را از نظر resource دارد.حال سوال اینجاست بر اساس چه مکانیزمی Node برای یک pod (براساس نیاز هایش) انتخاب میشود؟ Kuber-scheduler با طی دو مرحله node را برای pod انتخاب میکند.این دو مرحله عبارتند از:

- Filtering
- Scoring

در مرحله Filtering تمامی node ها از نظر resource و یک سری شرایط لازم مورد بررسی قرار میگرند یا به زبان ساده تر به یک سری سوالات در مورد نود پاسخ داده میشود.به عنوان مثال:

- آیا پورت درخواستی pod در نود مورد بررسی آزاد است؟
- آیا نود مورد بررسی میتواند جواب گوی resource های درخواستی pod ما باشد؟
- آیا نود ما به volumeدرخواستی از جانب pod دسترسی دارد؟

تمامی از شرایط ها در لینک زیر قابل دیدن است:

https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/

در صورتی که هر نودی از این مرحله به سلامت عبور کند به عنوان feasible node از آن یاد میشود و اگر هیچ nodeای در این مرحله انتخاب نشود Scheduling با مشکل مواجه شده است و pod ما هیچ نودی نخواهد داشت و به اصطلاح unscheduled میباشد تا زمانی که نودی برای آن انتخاب شود.در انتهای مرحله اول ما با لیستی از feasible نود ها طرف هستیم که وارد مرحله Scoring میشوند.

در مرحله ی Scoring کامپوننت Scheduler با اجرا یک سری function شروع به ارزش گذاری و امتیاز دهی به feasible نود ها میکند به عنوان مثال نودی که resource usageآن در حالت متعادل تر به سر میبرد امتیاز بهتری میگیرد و یا فرض کنید قرار است container ای را اجرا کنیم که قبلا هم وجود داشته است در این حالت نودی که قبلا میزبان این container بوده امتیاز بیشتری را از آن خود میکند.در انتهای این مرحله با لیست feasible node های ارزش گذاری شده طرف هستیم و نودی که

بیشترین امتیاز را دارد به عنوان میزبان pod انتخاب میشود و این تصمیم از طریق پروسه ای به نام binding به اطلاع API Server رسانده میشود.تمامی شرایط امتیاز دهی در همان لینک ذکر شده موجود است.

## Controller

در ابتدا باید دقت داشت که کامپوننت Controller خود از چندین Process تشکیل شده است اما برای تسهیل کار همه ی این Process ها در یک Controller یا بهتر بگوییم در یک Binary جمع آوری شده است. Controller در حقیقت به دنبال این است که آیا state فعلی همان state درخواستی میباشد؟به زبان ساده تر:آیا تعداد نسخه هایی که برای Pod A در نظر گرفته بودیم را الان بر روی نود های خود داریم؟آیا namespace که برای مجزا کردن pods های خود ساختیم دقیقا همانطور عمل میکند که میخواستیم؟تمامی این مسائل و نکات شبیه را Controller کنترل میکند.

Controller را میتوان همانند یک daemon در نظر گرفت که در یک loop بی نهایت قرار دارد و وظیفه او در این loop جمع کردن اطلاعات و فرستادن آنها به سمت API Serverمیباشد.اطلاعات جمع آوری شده در یک تعریف ساده همان حالت فعلی کلاستر (state of cluster) است که با حالت desired مقایسه میشود و در صورت هر گونه تفاوت تصمیماتی گرفته میشود.همانطور که در چند خط بالا گفته شد controller خود از controller ها کوچکتر ساخته شده است:

- Node Controller:یکی از اصلی ترین Controller ها میباشد دارای قوانین و رویکرد های مختلف در مورد سلامتی node ها میباشد.به عنوان مثال در kubernetes حالت NodeReady بهترین حالت یک node میباشد یعنی node آماده ی سرویس دهی میباشد.این controller وظیفه دارد به صورت مداوم حالت node ها را update کند و در صورتی که NodeReady به حالت دیگری تغییر پیدا کرد گزارش دهی کند.
- Replication Controller:مسولیت این controller در این سوال خلاصه میشود:آیا تعداد نسخه(replica)های موجود pod برابر با آن چیزی است که در desired state معلوم کردیم؟
- Endpoints Controller:در kubernetes از Endpoint به عنوان object هایی که در پشت صحنه ی یک سرویس فعالیت میکنند یاد میشود به عنوان مثال IP/PORT که برای یک سرویس مشخص میشود.وظیفه اصلی Endpoint Controller این است که مدام در حال بررسی این endpoint ها باشد تا از سلامت آنها اطمینان حاصل کند.

- Service Account & Token Controllers:کنترل کردن دسترسی ها به موجودیت در namespace های مختلف

در پایان باید در نظر داشت عمیق شدن بر روی هر کدام از این 4 کامپوننت بسیار وقت گیر و سخت است اما شکی نیست که هر چه بهتر با این کامپوننت ها آشنا شویم درک بهتری از چگونگی کار کردن kubernetes پیدا میکنیم و در زمان هایی که که با مشکل مواجه شویم سریع تر میتوانیم ریشه ی مشکل را پیدا کنیم.

Contact me on LinkedIn: https://www.linkedin.com/in/mohammad-ali-kamalian-7a3a72124/

## Resources


https://kubernetes.io/docs/concepts/overview/components/

https://kubernetes.io/docs/concepts/architecture/nodes/

https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/

https://medium.com/jorgeacetozi/kubernetes-master-components-etcd-api-server-controller-manager-and-scheduler-3a0179fc8186

https://medium.com/@dominik.tornow/kubernetes-api-server-part-i-3fbaf2138a31

https://kubernetes.io/docs/concepts/overview/kubernetes-api/

https://www.nakivo.com/blog/docker-vs-kubernetes/

https://thenewstack.io/kubernetes-an-overview/