一、三种方式：

1. 原生kubernetes的调度方式。官方文档《Running Spark on Kubernetes》；
2. 是由谷歌发起维护的管理Spark应用程序生命周期的Kubernetes插件，叫做SparkOperator。官方文档《spark-on-k8s-operator》；
3. 就是standalone的方式，就是在Kubernetes集群里面直接跑一套Spark的集群。这个没有官方文档参考；

二、分析：

第一种：

1. 该项目本来是由第三方在支持直到一年前被DEPRECATED，项目地址《Apache Spark enhanced with native Kubernetes scheduler back-end》。现在该项目的后续支持已经交由Spark官方在集成，但是各项在被DEPRECATED的那个项目中的功能还没完全集成官方的Spark应用中，比如：本地文件依赖管理的支持等。
2. 该方式的调度就没有master、worker的概念了，通过spark submit提交应用程序到kubernetes直接启动的是driver和excutor，具体启动instance的数量和要使用的资源根据配置而定。并且在也不会在kubernetes集群中一直运行着spark的进程，而是每次提交应用的时候才启动driver和excutor，当计算应用结束后excutor所在的pod立马被回收，而driver会被留下供查看日志等并等待kubernetes自动回收策略回收或者人工干预删除pod。
3. 提交应用和原生spark提交唯一参数区别就是master不一样，实例如下：

```
bin/spark-submit \
    --master k8s://https://10.0.10.197:6443 \
    --deploy-mode cluster \
    --name spark-SparkPi \
    --class org.apache.spark.examples.SparkPi \
    --conf spark.executor.instances=1 \
    --conf spark.kubernetes.container.image=kubespark/spark:2.3.2 \
    --conf spark.kubernetes.namespace=default \
    --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
    --conf spark.kubernetes.submission.waitAppCompletion=false \
    local:///opt/spark/examples/jars/spark-examples_2.11-2.3.2.java
```

4. 这里我们看到了，提交Spark的应用程序则需要我们在任何提交Spark应用的机器上要有一个编译好的Spark二进制包（不需要运行，只是借助它的spark-submit命令能力），并需要配置java环境供spark-submit运行。
5. 该方式需要编译Spark来支持kubernetes，当然官网打包编译好的文件是已经支持了的，如果自己编译就需要带上-Pkubernetes参数，可参考如下命令：

```
./build/mvn -Pkubernetes -Phadoop-2.7 -Dhadoop.version=2.7.3 -Phive -Phive-thriftserver -DskipTests clean
packag
```

6. 当然该方式也是需要我们打包一个支持Kubernetes的docker镜像的，并且需要将镜像分发到所有的Kubernetes机器节点或者上传到我们的私有镜像仓库中供kubernetes使用，打包镜像命令官网也提供了，参考如下命令：

```
./bin/docker-image-tool.sh -r kubespark -t 2.3.2 buil
```

8. 应用日志查看方式不再出现Spark Master UI页面，也没有History Server存储运行后的日志，而且Driver UI只能在Driver运行期间才能查看，一旦Driver生命周期结束也就不能查看了。但是我们可以通过Kubernetes来查看运行情况，比如通过kubectl logs driver-pod-name或者通过Kubernetes Dashboard从界面去查看。
9. 资源的分配直接交由kubernetes管理。

第二种：

1. 该方式需要在Kubernetes集群里面部署SparkOperator以便支持直接通过部署常规应用一样来部署Spark应用，这样我们就可以直接通过编写yaml文件来运行Spark应用，也不需要应用程序提交客户机上要有支持spark-submit的支持，只需要一个yaml文件。运行文件实例：

```
apiVersion: "sparkoperator.k8s.io/v1beta1"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: default
spec:
  type: Scala
  mode: cluster
  image: "kubespark/spark:2.4.2"
  imagePullPolicy: IfNotPresent
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "http://10.0.10.197:30010/spark/spark-examples_2.12-2.4.2.jar"
  sparkVersion: "2.4.2"
  restartPolicy:
    type: Never
  driver:
    cores: 0.1
    coreLimit: "200m"
    memory: "512m"
    labels:
      version: 2.4.2
    serviceAccount: spark
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 2.4.2
```

2. 但是该项目还处于beta阶段还在不断的完善。
3. 该方式除了需要再Kubernetes集群里面部署SparkOperator以外，同样需要第一种方式中的支持Kubernetes的Spark编译打包后的镜像，打包方式同第一种方式。
4. 该方式和第一种一样，没有Spark进程实现运行着等待应用提交，机制和第一种一样只运行driver和excutor，也没有master和worker的概念。
5. 资源的使用也是直接交由kubernetes管理。

第三种：

1. 这种方式和宿主机直接运行Spark集群几乎没什么大得区别，唯一方便的就是得力于Kubernetes的能力Worker可以很方便任性的伸缩。
2. 这种方式就是在Kubernetes集群中部署一套Spark的Standalone的集群，并且是一直运行着Master和N份Worker等待应用的提交。启动yaml配置如下：

```
apiVersion: v1
kind: Namespace
metadata:
  name: spark-cluster
  labels:
    name: spark-cluster

---

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: spark-master
  namespace: spark-cluster
  labels:
    name: spark-master
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: spark-master
    spec:
      hostname: spark-master
      containers:
      - name: spark-master
        image: spark:2.3.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 7077
        - containerPort: 8080
        - containerPort: 6066
        command:
         - "/bin/bash"
         - "-c"
        args :
         - './start-master.sh ; tail -f /dev/null'

---

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: spark-worker
  namespace: spark-cluster
  labels:
    name: spark-worker
spec:
  replicas: 3
  template:
    metadata:
      labels:
        name: spark-worker
    spec:
      containers:
      - name: spark-worker
        image: spark:2.3.2
        imagePullPolicy : IfNotPresent
        ports:
        - containerPort: 8081
        command:
         - "/bin/bash"
         - "-c"
        args :
         - './start-worker.sh ; tail -f /dev/null'

---

kind: ReplicationController
apiVersion: v1
metadata:
  name: spark-ui-proxy
  namespace: spark-cluster
spec:
  replicas: 1
  selector:
    name: spark-ui-proxy
  template:
    metadata:
      labels:
        name: spark-ui-proxy
    spec:
      hostname: spark-ui-proxy
      containers:
        - name: spark-ui-proxy
          image: spark-ui-proxy:1.0
          imagePullPolicy : IfNotPresent
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: 100m
          args:
            - spark-master:8080
          livenessProbe:
              httpGet:
                path: /
                port: 80
              initialDelaySeconds: 120
              timeoutSeconds: 5

---

apiVersion: v1
kind: Service
metadata:
  name: spark-master
  namespace: spark-cluster
  labels:
    name: spark-master
spec:
  clusterIP: None
  ports:
  - name: webui
    port: 8080
    targetPort: 8080
  - name: spark
    port: 7077
    targetPort: 7077
  - name: rest
    port: 6066
    targetPort: 6066
  selector:
    name: spark-master

---

kind: Service
apiVersion: v1
metadata:
  name: spark-worker
  namespace: spark-cluster
spec:
  ports:
  - name: workerui
    port: 8081
    targetPort: 8081
  selector:
    name: spark-worker

---

kind: Service
apiVersion: v1
metadata:
  name: spark-ui-proxy
  namespace: spark-cluster
spec:
  ports:
  - port: 80
    targetPort: 80
  selector:
    name: spark-ui-proxy

---

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: spark-ingress
  namespace: spark-cluster
spec:
  rules:
  - http:
      paths:
      - path: /sparkui
        backend:
          serviceName: spark-ui-proxy
          servicePort: 80
      - path: /proxy
        backend:
          serviceName: spark-ui-proxy
          servicePort: 80
      - path: /app
        backend:
          serviceName: spark-ui-proxy
          servicePort: 80
```

3. 该种方式唯一由Kubernetes接管的资源就是Master和Work可用资源，就是启动Master和Worker的时候给它们配置的资源，但是我们提交的应用程序需要的资源仍然是由Master来管控的，当然Master能用多少又是Kubernetes来管控的。
4. 该种运行方式在提交应用程序在deploy-mode使用cluster的时候由于Worker运行Excutor的时候使用的计算机的hostname作为Driver URL使用，spark集群所在计算机的hostname需要事先配置好，但是搬到Kubernetes上后由于Worker节点是通过Kubernetes的pod启动的多副本Deployment，hostname是随机生成不可预知不能提前配置导致exceutor不能启动。经多番折腾通过Kubernetes暂不能解决该问题，最后只能通过修改spark源码通过ip方式可以解决。但是影响范围需要张奇确认（论对spark源码熟悉度非张奇莫属了）。
5. 提交应用方式也是通过spark-submit方式，和宿主机部署集群提交方式一样。实例：

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://spark-master:7077 \
  --deploy-mode client \
  --executor-memory 1G \
  --total-executor-cores 2 \
  /spark/examples/jars/spark-examples_2.11-2.3.2.jar \
  1000

./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://spark-master:6066 \
  --deploy-mode cluster \
  --executor-memory 1G \
  --total-executor-cores 2 \
  /spark/examples/jars/spark-examples_2.11-2.3.2.jar \
  1000
```

6. 此方式我们也需要将spark打包成docker镜像来运行，打包方式和上两种方式对kubernetes原始资源支持需要的镜像包有所不同，就是将宿主机运行方式的命令使用在镜像里面而已。
7. 日志的查看和宿主机运行也没缺别，不管是Master UI还是History Server。

三、对比：

1. 第一种方式有官方支持，应该会越来越好用，且支持原生的Kubernetes资源调度；
2. 第二种方式也是大厂支持，基于第一种方式和Kubernetes的深度集成（建议采用这种方式）；
3. 第三种standalone方式，只是把Spark集群部署搬到了Kubernetes不支持Kubernetes原生资源调度，只是和只是集群交由Kubernetes管控而已；