

### Step 1: Clone the Repository

```
```bash
git clone https://github.com/ndeepakprasanth/dapr-eks-pubsub-ready.git
cd dapr-eks-pubsub-ready
```

### Step 2: Set execute permissions

```
chmod +x oneclick.sh
chmod +x test.sh
chmod +x scripts/*.sh
```

### Step 3: One-Click deployment

```
# Replace with your AWS Account ID
ACCOUNT_ID=946248011760 ./oneclick.sh
```

### Step 4: Verify deployment

```
# Check pods (should show 2/2 Ready)
kubectl -n dapr-apps get pods -o wide
# Check services
kubectl -n dapr-apps get svc
# Check Dapr components
kubectl -n dapr-apps get components
```

### Step 5: Test Pub/Sub functionality

```
./test.sh
```

### Step 6: View ECR repositories

```
aws ecr describe-repositories --region us-east-1
```

### Step 7: Clean up when completed

```
./destroy.sh
```

#### 1. Running pods:

```
423946@AMBGB000622 dapr-eks-pubsub-ready % kubectl -n dapr-apps get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE                                NOMINATED NODE   READINESS GATES
orderservice-974558c9b-q88hz        2/2     Running   0           4m41s  192.168.47.173  ip-192-168-36-44.ec2.internal      <none>           <none>
productservice-787cc7d9c7-8ztmz     2/2     Running   0           4m41s  192.168.26.113  ip-192-168-13-87.ec2.internal      <none>           <none>
423946@AMBGB000622 dapr-eks-pubsub-ready %
```

#### 2. Services:

```
423946@AMBGB000622 dapr-eks-pubsub-ready % kubectl -n dapr-apps get svc
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
orderservice                        ClusterIP    10.100.48.232    <none>            8090/TCP          48m
orderservice-dapr                   ClusterIP    None             <none>            80/TCP,50001/TCP,50002/TCP,9090/TCP  48m
productservice                      ClusterIP    10.100.230.157   <none>            8080/TCP          48m
productservice-dapr                 ClusterIP    None             <none>            80/TCP,50001/TCP,50002/TCP,9090/TCP  48m
```

#### 3. Dapr components

```
423946@AMBGB000622 dapr-eks-pubsub-ready % kubectl -n dapr-apps get components
NAME                AGE
snssqs-pubsub       6m15s
```

#### 4. Test Pub/Sub

```
423946@AMBG000622 dapr-eks-pubsub-ready % kubectl -n dapr-apps run test-curl --rm -i --restart=Never --image=curlimages/curl:8.10.1 -- curl -X POST http://productservice:8080/publish -H 'Content-type: application/json' -d '{"orderId": 999, "item": "laptop", "price": 199.99}'

If you don't see a command prompt, try pressing enter.
warning: couldn't attach to pod/test-curl, falling back to streaming logs: unable to upgrade connection: container test-curl not found in pod test-curl_dapr-apps
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           % Done    0     0      0         0      0      0      0      0
100 1400 100 1350 100    50 34119 1263    --:--:-- --:--:-- --:--:-- 35897
{"ok":false,"error":{"Request failed with status code 500","details":{"errorCode":"ERR_PUBSUB_PUBLISH_MESSAGE","message":"error when publishing to topic orders in pubsub sns
sqs-pubsub: error publishing to topic: orders with topic ARN : operation error SNS: Publish, https response error StatusCode: 400, RequestID: 5d77f1b3-72a7-5cb5-82b6-90704f
1f5006, InvalidParameter: Invalid parameter: TopicArn Reason: An ARN must have at least 6 elements, not 1","details":{"@type":"type.googleapis.com/google.rpc.ResourceInfo
","description":"error when publishing to topic orders in pubsub sns","resource_type":"pubsub"},"@type":"type.googleapis.com/google.rpc.ErrorInfo","domain":"dapr.io","metadata":{"error":"error pub
lishing to topic: orders with topic ARN : operation error SNS: Publish, https response error StatusCode: 400, RequestID: 5d77f1b3-72a7-5cb5-82b6-90704f1f5006, InvalidParamete
rs: Invalid parameter: TopicArn Reason: An ARN must have at least 6 elements, not 1","topic":"orders"},"reason":"DAPR_PUBSUB_PUBLISH_MESSAGE"}}}pod "test-curl" deleted
423946@AMBG000622 dapr-eks-pubsub-ready %
```

#### 5. ECR repositories with pushed images

```
423946@AMBG000622 dapr-eks-pubsub-ready % aws ecr describe-repositories --region us-east-1 --profile Deepak
{
  "repositories": [
    {
      "repositoryArn": "arn:aws:ecr:us-east-1:946248011760:repository/orderservice",
      "registryId": "946248011760",
      "repositoryName": "orderservice",
      "repositoryUri": "946248011760.dkr.ecr.us-east-1.amazonaws.com/orderservice",
      "createdAt": "2025-12-22T11:41:09.817000+00:00",
      "imageTagMutability": "MUTABLE",
      "imageScanningConfiguration": {
        "scanOnPush": false
      },
      "encryptionConfiguration": {
        "encryptionType": "AES256"
      }
    },
    {
      "repositoryArn": "arn:aws:ecr:us-east-1:946248011760:repository/productservice",
      "registryId": "946248011760",
      "repositoryName": "productservice",
      "repositoryUri": "946248011760.dkr.ecr.us-east-1.amazonaws.com/productservice",
      "createdAt": "2025-12-22T11:41:06.238000+00:00",
      "imageTagMutability": "MUTABLE",
      "imageScanningConfiguration": {
        "scanOnPush": false
      },
      "encryptionConfiguration": {
        "encryptionType": "AES256"
      }
    }
  ]
}
423946@AMBG000622 dapr-eks-pubsub-ready %
```

## 6. Bedrock analysis

```
⇒ Bedrock GenAI Analysis Prompts for Assignment

📄 Copy these prompts to Amazon Bedrock Console:

🔗 TELEMETRY ANALYSIS PROMPT:
=====
Given the attached Node.js microservices (ProductService and OrderService) running on EKS with Dapr sidecars for pub/sub messaging via AWS SNS/SQS, suggest missing telemetry points:

- Custom application logs for business events
- Distributed trace spans and attributes
- Metrics for publish latency, delivery success/failure rates
- Dead letter queue monitoring
- Dapr sidecar performance metrics

Output OpenTelemetry configuration snippets for Node.js applications and Kubernetes manifests.

🔗 RESILIENCY ANALYSIS PROMPT:
=====
Recommend retry, backoff, and circuit-breaker policies for event-driven architecture using:

- Dapr pub/sub with AWS SNS/SQS
- EKS microservices with potential network failures
- Application-level idempotency handling

Include:

- Dapr resiliency policies YAML configuration
- Sample deduplication logic for Node.js
- Error handling patterns for pub/sub failures

🔗 SECURITY & PERFORMANCE ANALYSIS PROMPT:
=====
Analyze the following artifacts for security and performance issues:

- Dockerfiles (Node.js Alpine-based containers)
- Kubernetes Deployment YAML (ProductService, OrderService)
- Dapr SNS/SQS component configuration

Flag issues like:

- Root user usage in containers
- Missing resource requests/limits
- Absent liveness/readiness probes
- Message concurrency settings
- Security contexts and capabilities

Provide specific fixes and improved configurations.

🔗 SCALING PATTERNS PROMPT:
=====
For SQS + Dapr pub/sub on Amazon EKS, propose:

- Horizontal Pod Autoscaling based on SQS queue length using KEDA
- Dapr bulkSubscribe settings for high throughput
- Resource optimization for cost-effective scaling
- Monitoring and alerting for scaling events

Include complete Kubernetes manifests for KEDA ScaledObject and HPA configurations.
```

## 7. Stress test output

```
423946@AMBG800622 dapr-eks-pubsub-ready% echo "=== STRESS TEST DEMONSTRATION ==="
echo "Sending multiple concurrent requests..."
for i in {1..5}; do
  kubectl -n dapr-apps run load-test-$i --restart=Never --image=curlimages/curl:8.10.1 --command -- curl -X POST http://productservice:8080/publish -H 'Content-Type: application/json' -d '{"orderId": $i, "item": "load-test-$i", "price": ${RANDOM % 1000}}' &
done
echo "Waiting for completion..."
sleep 8
echo ""
echo "=== STRESS TEST RESULTS ==="
kubectl -n dapr-apps get pods | grep load-test
echo ""
echo "=== CLEANUP ==="
kubectl -n dapr-apps delete pods -l run --field-selector=status.phase=Succeeded
=== STRESS TEST DEMONSTRATION ===
Sending multiple concurrent requests...
[2] 7386
[3] 7387
[4] 7388
[5] 7389
[6] 7390
Waiting for completion...
pod/load-test-4 created
pod/load-test-4 created
[6] + done      kubectl -n dapr-apps run load-test-$i --restart=Never --command -- curl -X
[5] + done      kubectl -n dapr-apps run load-test-$i --restart=Never --command -- curl -X
pod/load-test-1 created
[2] done       kubectl -n dapr-apps run load-test-$i --restart=Never --command -- curl -X
pod/load-test-3 created
[4] + done      kubectl -n dapr-apps run load-test-$i --restart=Never --command -- curl -X
pod/load-test-2 created
[3] + done      kubectl -n dapr-apps run load-test-$i --restart=Never --command -- curl -X

=== STRESS TEST RESULTS ===
load-test-1          0/1    Completed    0      8s
load-test-2          0/1    Completed    0      8s
load-test-3          0/1    Completed    0      8s
load-test-4          0/1    Completed    0      8s
load-test-5          0/1    Completed    0      8s

=== CLEANUP ===
pod "load-test-1" deleted
pod "load-test-2" deleted
pod "load-test-3" deleted
pod "load-test-4" deleted
pod "load-test-5" deleted
```

## 8. Logs screenshot:

```
423946@AMBG00622 dapr-eks-pubsub-ready % echo "=== ARCHITECTURE VERIFICATION ==="
echo ""
EKS Cluster Nodes:
kubectl get nodes -o wide
echo ""
Dapr System Pods:
kubectl -n dapr-system get pods
echo ""
Application Logs (ProductService):
kubectl -n dapr-apps logs deploy/productservice --tail=5
echo ""
Application Logs (OrderService):
kubectl -n dapr-apps logs deploy/orderservice --tail=5
echo ""
=== ARCHITECTURE VERIFICATION ===

EKS Cluster Nodes:
NAME                                STATUS  ROLES  AGE  VERSION  INTERNAL-IP  EXTERNAL-IP  OS-IMAGE  KERNEL-VERSION
CONTAINER-RUNTIME
ip-192-168-13-87.ec2.internal        Ready  <none>  164m  v1.32.9-eks-ecaa3a6  192.168.13.87  100.49.41.36  Amazon Linux 2023.9.20251208  6.1.158-180.294.amzn2023.x86_64
ip-192-168-36-44.ec2.internal        Ready  <none>  164m  v1.32.9-eks-ecaa3a6  192.168.36.44  54.196.115.105  Amazon Linux 2023.9.20251208  6.1.158-180.294.amzn2023.x86_64

Dapr System Pods:
NAME                                READY  STATUS  RESTARTS  AGE
dapr-operator-69845db889-4bmp6      1/1    Running  0          162m
dapr-placement-server-0             1/1    Running  0          162m
dapr-scheduler-server-0             1/1    Running  0          162m
dapr-scheduler-server-1             1/1    Running  0          162m
dapr-scheduler-server-2             1/1    Running  0          162m
dapr-sentry-78f477df9c-wvg7p        1/1    Running  0          162m
dapr-sidecar-injector-6f774f9bfc-c54cw 1/1    Running  0          162m

Application Logs (ProductService):
Defaulted container "productservice" out of: productservice, daprd
metadata: [Object],
reason: 'DAPR_PUBSUB_PUBLISH_MESSAGE'
}
}

Application Logs (OrderService):
Defaulted container "orderservice" out of: orderservice, daprd
OrderService listening on 8090
```

## 9. Bedrock analysis

```
423946@AMBG00622 dapr-eks-pubsub-ready % echo "BEDROCK GENAI ANALYSIS RESULTS:"
echo ""
cat bedrock-log-analysis.txt
BEDROCK GENAI ANALYSIS RESULTS:

The provided content is a list of actions to be taken to resolve the issues mentioned in the logs.

1. Root cause analysis: Investigate the root cause of the authorization failures and invalid TopicArn format errors. This may involve reviewing the IAM policies, roles, and permissions associated with the AWS resources involved in the microservices and SNS topic creation.

2. IAM policy fixes: Update the IAM policies to grant the necessary permissions to Dapr and SNS to create and publish messages to the SNS topic. This may involve adding appropriate actions to the IAM policies or adjusting existing policies.

3. Application error handling: Implement error handling logic in the Dapr microservices to handle any errors that occur during SNS topic creation or message publishing. This may involve catching exceptions, logging errors, and returning appropriate error responses to the caller.

4. Monitoring recommendations: Implement monitoring of the Dapr microservices and SNS topic to detect any issues that may arise during the stress testing process. This may involve setting up metrics and alerts to track key performance indicators and identify potential problems before they become critical.

5. Testing and validation: Conduct thorough testing and validation of the Dapr microservices and SNS topic configuration to ensure that they are functioning as expected. This may involve running performance tests, load tests, and stress tests to simulate real-world scenarios and identify any bottlenecks or issues.

6. Documentation: Update the documentation to include the IAM policy fixes, application error handling, and monitoring recommendations. This will help other developers and stakeholders understand how to manage the microservices and SNS topic effectively.

7. Collaboration: Work closely with the development, operations, and security teams to ensure that all changes to the microservices and SNS topic are properly coordinated and documented. This will help to ensure that any changes are made in a controlled and secure manner.

8. Continuous improvement: Continuously review and improve the microservices and SNS topic configuration based on feedback and performance metrics. This may involve experimenting with different configurations, optimizing resource usage, and implementing new features or functionality.

By following these steps, you can improve the reliability and performance of the microservices and SNS topic and ensure that they are able to handle the stress testing requirements of your application.
```