````
# MZ2POL-00: SDK Management Zone Analysis Tool

> **Series:** MZ2POL | **Notebook:** 1 of 8 | **Created:** December 2025

> **Purpose:** Query Management Zone configurations via the Dynatrace SDK,
analyze entity assignments, and assess security context coverage to support
migration planning.

## What This Notebook Does

1. **Export all MZ configurations** - Get all MZ rules in a flat table format
via Settings API
2. **Analyze MZ rule patterns** - Understand what types of rules are in use
3. **Check entity security context coverage** - Find entities missing
`dt.security_context`
4. **Identify most-used MZs** - Prioritize migration based on usage
5. **Find tag patterns** - Discover common tag structures for segment
creation
6. **Kubernetes namespace analysis** - K8s namespaces for segment variables
7. **Host group analysis** - Host groups for vertical MZ migration
8. **Migration readiness summary** - Combined view of readiness indicators

---

## 1. Export All Management Zone Configurations

This TypeScript function queries all MZs via the Settings API and flattens
their rules into an analyzable table.

```python
import { settingsObjectsClient } from "@dynatrace-sdk/client-classic-
environment-v2";

// Type definitions
interface MzAttributeRuleCondition {
  key: string;
  operator: string;
  tag?: string;
  stringValue?: string;
  caseSensitive?: boolean;
}

interface MzAttributeRule {
  entityType: string;
  serviceToHostPropagation?: boolean;
  serviceToPGPropagation?: boolean;
  conditions?: MzAttributeRuleCondition[];
````
````

```
}

interface MzRule {
  enabled: boolean;
  type: string;
  entitySelector?: string;
  attributeRule: MzAttributeRule[];
}

interface MzDefinition {
  name: string;
  description?: string;
  rules: MzRule[];
}

// Fetch all MZs with pagination
async function getAllMZs(nextPageKey?: string, mzList: MzDefinition[] = []):
Promise {
  const config: any = nextPageKey
    ? { nextPageKey }
    : { schemaIds: "builtin:management-zones", pageSize: 100, fields:
"objectId,value" };

  const resp = await settingsObjectsClient.getSettingsObjects(config);
  mzList.push(...resp.items.map(elem => elem.value as MzDefinition));

  return resp.nextPageKey ? getAllMZs(resp.nextPageKey, mzList) : mzList;
}

// Flatten rules into records
function flattenRules(mzList: MzDefinition[]): any[] {
  const records: any[] = [];

  mzList.forEach(mz => {
    mz.rules?.forEach(rule => {
      if (rule.attributeRule) {
        const attr = rule.attributeRule;
        const base = {
          mzName: mz.name,
          description: mz.description || "",
          ruleType: rule.type,
          ruleEnabled: rule.enabled,
          entityType: attr.entityType || "",
          propagateToHost: attr.serviceToHostPropagation || false,
          propagateToPG: attr.serviceToPGPropagation || false
        };

        if (attr.conditions?.length) {
```

```
          attr.conditions.forEach(c => {
            records.push({
              ...base,
              conditionKey: c.key,
              operator: c.operator,
              value: c.stringValue || c.tag || "",
              caseSensitive: c.caseSensitive || false
            });
          });
        } else {
          records.push({ ...base, conditionKey: "", operator: "", value: "",
caseSensitive: false });
        }
      }

      if (rule.type === "SELECTOR") {
        records.push({
          mzName: mz.name,
          description: mz.description || "",
          ruleType: "SELECTOR",
          ruleEnabled: rule.enabled,
          entityType: "",
          propagateToHost: false,
          propagateToPG: false,
          conditionKey: "entitySelector",
          operator: "SELECTOR",
          value: rule.entitySelector || "",
          caseSensitive: false
        });
      }
    });
  });

  return records;
}

export default async function() {
  const mzList = await getAllMZs();
  return flattenRules(mzList);
}
```

---

2. MZ Rule Pattern Analysis

Analyze what types of MZ rules are in use to understand migration complexity.

```python
import { settingsObjectsClient } from "@dynatrace-sdk/client-classic-environment-v2";

interface MzRule {
  enabled: boolean;
  type: string;
  entitySelector?: string;
  attributeRule?: { entityType: string; conditions?: { key: string; operator: string }[] }[];
}

interface MzDefinition {
  name: string;
  rules: MzRule[];
}

async function getAllMZs(nextPageKey?: string, mzList: MzDefinition[] = []): Promise {
  const config: any = nextPageKey
    ? { nextPageKey }
    : { schemaIds: "builtin:management-zones", pageSize: 100, fields: "objectId,value" };
  const resp = await settingsObjectsClient.getSettingsObjects(config);
  mzList.push(...resp.items.map(elem => elem.value as MzDefinition));
  return resp.nextPageKey ? getAllMZs(resp.nextPageKey, mzList) : mzList;
}

export default async function() {
  const mzList = await getAllMZs();

  // Count patterns
  const stats = {
    totalMZs: mzList.length,
    totalRules: 0,
    byRuleType: {} as Record,
    byEntityType: {} as Record,
    byConditionKey: {} as Record,
    byOperator: {} as Record,
    selectorRules: 0,
    meRules: 0,
    tagBasedRules: 0,
    nameBasedRules: 0
  };

  mzList.forEach(mz => {
    mz.rules?.forEach(rule => {
      stats.totalRules++;
```

```
      stats.byRuleType[rule.type] = (stats.byRuleType[rule.type] || 0) + 1;

      if (rule.type === "SELECTOR") {
        stats.selectorRules++;
      } else if (rule.type === "ME" && rule.attributeRule) {
        stats.meRules++;
        const attr = rule.attributeRule;
        stats.byEntityType[attr.entityType] =
(stats.byEntityType[attr.entityType] || 0) + 1;

        attr.conditions?.forEach(c => {
          stats.byConditionKey[c.key] = (stats.byConditionKey[c.key] || 0) +
1;
          stats.byOperator[c.operator] = (stats.byOperator[c.operator] || 0)
+ 1;

          if (c.key.includes("TAGS")) stats.tagBasedRules++;
          if (c.key.includes("NAME")) stats.nameBasedRules++;
        });
      }
    });
  });

  // Convert to table format
  const results = [
    { category: "Summary", metric: "Total Management Zones", count:
stats.totalMZs },
    { category: "Summary", metric: "Total Rules", count: stats.totalRules },
    { category: "Summary", metric: "Avg Rules per MZ", count:
Math.round(stats.totalRules / stats.totalMZs * 10) / 10 },
    { category: "Rule Type", metric: "SELECTOR rules", count:
stats.selectorRules },
    { category: "Rule Type", metric: "ME (attribute) rules", count:
stats.meRules },
    { category: "Pattern", metric: "Tag-based rules", count:
stats.tagBasedRules },
    { category: "Pattern", metric: "Name-based rules", count:
stats.nameBasedRules }
  ];

  // Add top entity types
  Object.entries(stats.byEntityType)
    .sort((a, b) => b[1] - a[1])
    .slice(0, 10)
    .forEach(([type, count]) => {
      results.push({ category: "Entity Type", metric: type, count });
    });
```

```
  // Add top condition keys
  Object.entries(stats.byConditionKey)
    .sort((a, b) => b[1] - a[1])
    .slice(0, 10)
    .forEach(([key, count]) => {
      results.push({ category: "Condition Key", metric: key, count });
    });

  return results;
}
```

---

3. Entity Security Context Coverage

Check how many entities have `dt.security_context` set vs those relying only on `managementZones`.

Service Coverage

```python
fetch dt.entity.service
| summarize
    total = count(),
    withSecurityContext = countIf(isNotNull(dt.security_context)),
    withMZs = countIf(isNotNull(managementZones)),
    noAccess = countIf(isNull(dt.security_context) AND
isNull(managementZones))
| fieldsAdd
    securityContextPercent = round(100.0 * withSecurityContext / total,
decimals: 2),
    mzPercent = round(100.0 * withMZs / total, decimals: 2)
```

Host Coverage

```python
fetch dt.entity.host
| summarize
    total = count(),
    withSecurityContext = countIf(isNotNull(dt.security_context)),
    withMZs = countIf(isNotNull(managementZones))
| fieldsAdd
    securityContextPercent = round(100.0 * withSecurityContext / total,
decimals: 2),
    mzPercent = round(100.0 * withMZs / total, decimals: 2)
```

````
---

## 4. Entity Counts by Management Zone

See how many entities are assigned to each MZ to prioritize migration.

### Services per Management Zone

```python
fetch dt.entity.service
| expand mz = managementZones
| filter isNotNull(mz)
| summarize entityCount = count(), by: {managementZone = mz}
| sort entityCount desc
| limit 30
```

---

## 5. Tag Pattern Analysis

Discover common tag patterns that could be used for segment creation.

### Most Common Host Tags

```python
fetch dt.entity.host
| expand tag = tags
| filter isNotNull(tag)
| summarize count = count(), by: {tag}
| sort count desc
| limit 50
```

### Tag Key Frequency

Extract key from `key:value` format to see which tag keys are most common.

```python
fetch dt.entity.host
| expand tag = tags
| filter isNotNull(tag)
| filter contains(tag, ":")
| parse tag, "LD:tagKey ':' LD:tagValue"
| summarize count = count(), uniqueValues = countDistinct(tagValue), by:
{tagKey}
| sort count desc
````
````

````
| limit 30
```

---

## 6. Kubernetes Namespace Analysis

K8s namespaces are a common source for segment variables.

```python
fetch dt.entity.cloud_application_namespace
| fields namespace = entity.name, id
| sort namespace asc
| limit 30
```

---

## 7. Host Group Analysis

Host groups are key for vertical MZ migration (environment-based access control).

```python
fetch dt.entity.host_group
| fields hostGroup = entity.name, id
```

---

## 8. Migration Readiness Summary

Combined view of migration readiness indicators.

```python
import { settingsObjectsClient } from "@dynatrace-sdk/client-classic-environment-v2";
import { queryExecutionClient } from "@dynatrace-sdk/client-query";

async function getMZCount(): Promise {
  const resp = await settingsObjectsClient.getSettingsObjects({
    schemaIds: "builtin:management-zones",
    pageSize: 1,
    fields: "totalCount"
  });
  return resp.totalCount || 0;
}
````

```
async function runDQL(query: string): Promise {
  const resp = await queryExecutionClient.queryExecute({
    body: {
      query,
      requestTimeoutMilliseconds: 60000,
      maxResultRecords: 10
    }
  });
  return resp.result?.records || [];
}

export default async function() {
  const mzCount = await getMZCount();

  // Get service coverage
  const serviceCoverage = await runDQL(`
    fetch dt.entity.service
    | summarize
        total = count(),
        withSecContext = countIf(isNotNull(dt.security_context)),
        withMZ = countIf(isNotNull(managementZones))
  `);

  // Get host coverage
  const hostCoverage = await runDQL(`
    fetch dt.entity.host
    | summarize
        total = count(),
        withSecContext = countIf(isNotNull(dt.security_context))
  `);

  const svc = serviceCoverage[0] || {};
  const host = hostCoverage[0] || {};

  return [
    { category: "Management Zones", metric: "Total MZs to migrate", value:
mzCount, status: mzCount > 50 ? "HIGH" : "OK" },
    { category: "Services", metric: "Total services", value: svc.total || 0,
status: "INFO" },
    { category: "Services", metric: "With security_context", value:
svc.withSecContext || 0, status: (svc.withSecContext / svc.total) > 0.8 ?
"GOOD" : "NEEDS_WORK" },
    { category: "Services", metric: "With MZ assignment", value: svc.withMZ
|| 0, status: "INFO" },
    { category: "Hosts", metric: "Total hosts", value: host.total || 0,
status: "INFO" },
    { category: "Hosts", metric: "With security_context", value:
host.withSecContext || 0, status: (host.withSecContext / host.total) > 0.8 ?
```

````
"GOOD" : "NEEDS_WORK" }
  ];
}
```

---

## Next Steps

### Based on the analysis above:

1. **High MZ count?** -> Prioritize by usage (query
`dt.sfm.server.management_zones.queries_counter`)
2. **Low security_context coverage?** -> Plan host tag deployment or
OpenPipeline enrichment
3. **Many tag-based MZ rules?** -> Map to segment filters using
`matchesValue(tags, "key:value")`
4. **Many SELECTOR rules?** -> Review entity selectors for segment conversion
5. **Host groups in use?** -> Use `dt.host_group.id` in boundaries for
vertical MZ migration

### Migration Priority:

| Priority | Criteria | Action |
|----------|----------|--------|
| 1 | MZs with >100 entities | Migrate first - highest impact |
| 2 | Tag-based MZ rules | Easy segment conversion |
| 3 | Host group-based MZs | Use boundary on `dt.host_group.id` |
| 4 | Entity selector MZs | Review and convert to segments |
| 5 | Complex/nested MZs | Manual analysis required |

---

## Related Notebooks

- **MZ2POL-01**: Introduction - Why Migrate
- **MZ2POL-03**: Assessment and Planning
- **MZ2POL-06**: Migration Execution
- **MZ2POL-07**: Validation and Troubleshooting

## Documentation

- [Management Zones Documentation]
(https://docs.dynatrace.com/docs/manage/identity-access-
management/permission-management/management-zones)
- [Upgrade from RBAC to IAM Policies]
(https://docs.dynatrace.com/docs/manage/identity-access-
management/permission-management/manage-user-permissions-
````
````

```
policies/advanced/migrate-roles)
- [Grant Access to Entities with Security Context]
(https://docs.dynatrace.com/docs/manage/identity-access-management/use-
cases/access-security-context)

---

*MZ2POL-00: SDK Management Zone Analysis Tool - Part of the MZ to
Policies/Boundaries/Segments Series*
```